Advanced Data Structures and Algorithm Analysis

丁尧相
浙江大学

Fall & Winter 2025
Lecture 5

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
- Take-home messages

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
- Take-home messages

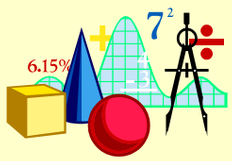
Recursively:

Divide the problem into a number of sub-problems

Conquer the sub-problems by solving them recursively

Combine the solutions to the sub-problems into the solution for the original problem

General recurrence: $T(N) = aT(N/b) + f(N)$



Cases solved by divide and conquer

- The maximum subsequence sum – the O($N \log N$) solution
- Tree traversals – O(N)
- Mergesort and quicksort – O($N \log N$)

Maximum Sequence of Sum

〖Example〗 Given (possibly negative) integers $A_1, A_2, \ldots, A_N$, find the maximum value of $\sum_{k=i}^{j} A_k$.

Algorithm 1

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/  MaxSum = 0;   /* initialize the maximum sum */
/* 2*/  for( i = 0; i < N; i++ )  /* start from A[ i ] */
/* 3*/      for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 4*/          ThisSum = 0;
/* 5*/          for( k = i; k <= j; k++ )
/* 6*/              ThisSum += A[ k ];  /* sum from A[ i ] to A[ j ] */
/* 7*/          if ( ThisSum > MaxSum )
/* 8*/              MaxSum = ThisSum;  /* update max sum */
            } /* end for-j and for-i */
/* 9*/  return  MaxSum;
}
```

$$T(N) = O(N^3)$$

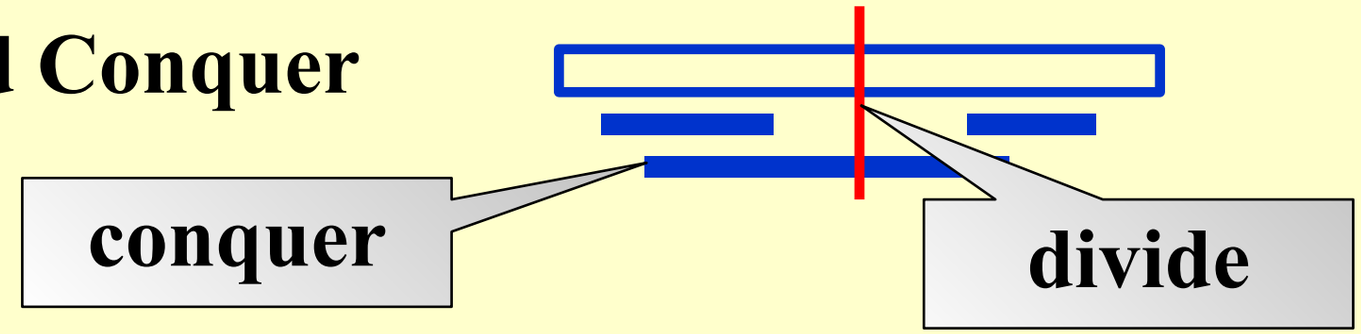
Algorithm 2

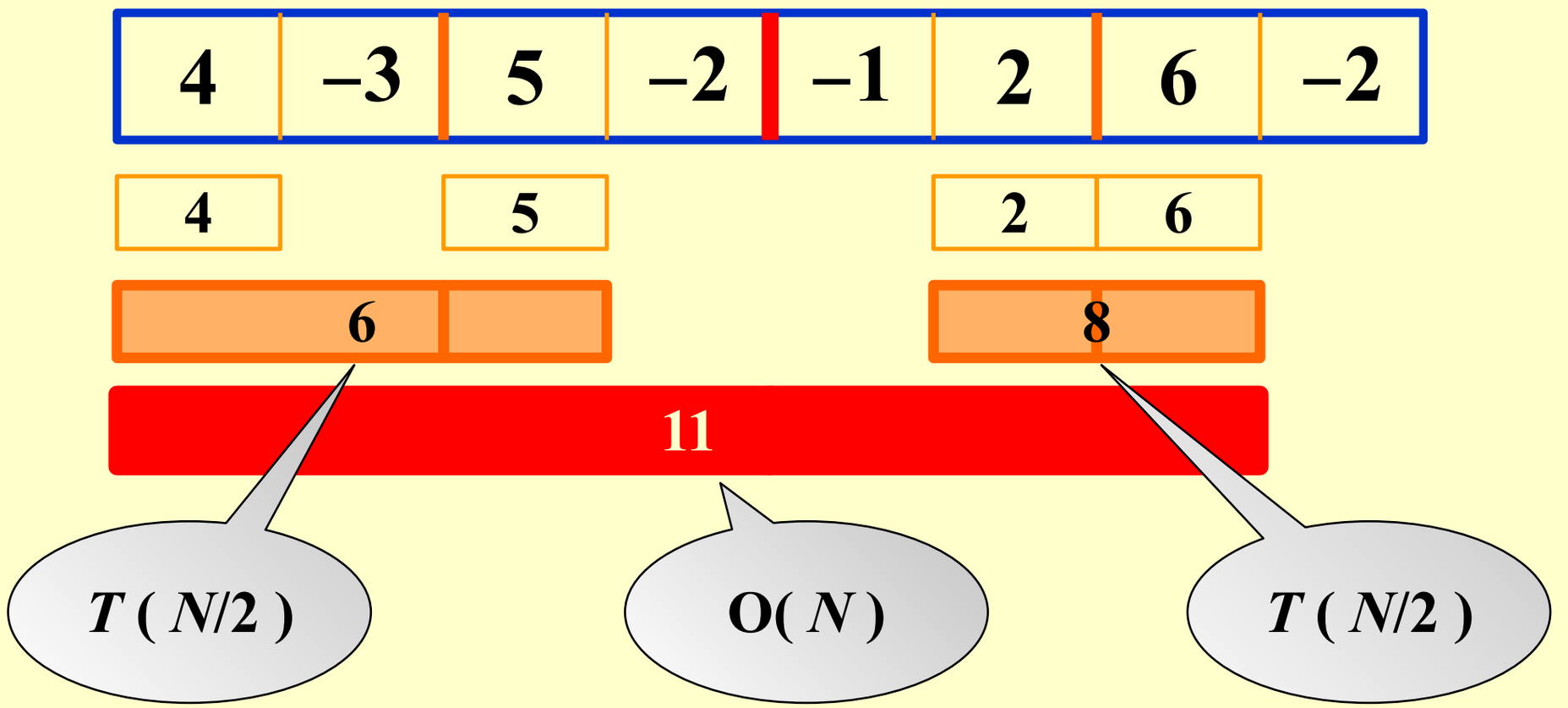
```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
          int  ThisSum,  MaxSum,  i,  j;
/* 1*/    MaxSum = 0;   /* initialize the maximum sum */
/* 2*/    for( i = 0; i < N; i++ )  {   /* start from A[ i ] */
/* 3*/         ThisSum = 0;
/* 4*/         for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 5*/              ThisSum += A[ j ];  /* sum from A[ i ] to A[ j ] */
/* 6*/              if ( ThisSum > MaxSum )
/* 7*/                   MaxSum = ThisSum;  /* update max sum */
              }  /* end for-j */
          }  /* end for-i */
/* 8*/    return  MaxSum;
}
```

$$T(N) = O(N^2)$$

Algorithm 3

Divide and Conquer





$T(N) = 2\,T(N/2) + cN, \qquad T(1) = O(1)$

$= 2\,[2\,T(N/2^2) + cN/2] + cN$

$= 2^k\,O(1) + ckN \qquad$ where $N/2^k =$

$= O(N \log N)$

Also true for $N \neq 2^k$

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into two subproblems of size $n/2$. ← $O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. ← $O(n)$ time

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $O(n \log n)$.

Common situation.
Not always the case.

attributed to Julius Caesar

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
- Take-home messages

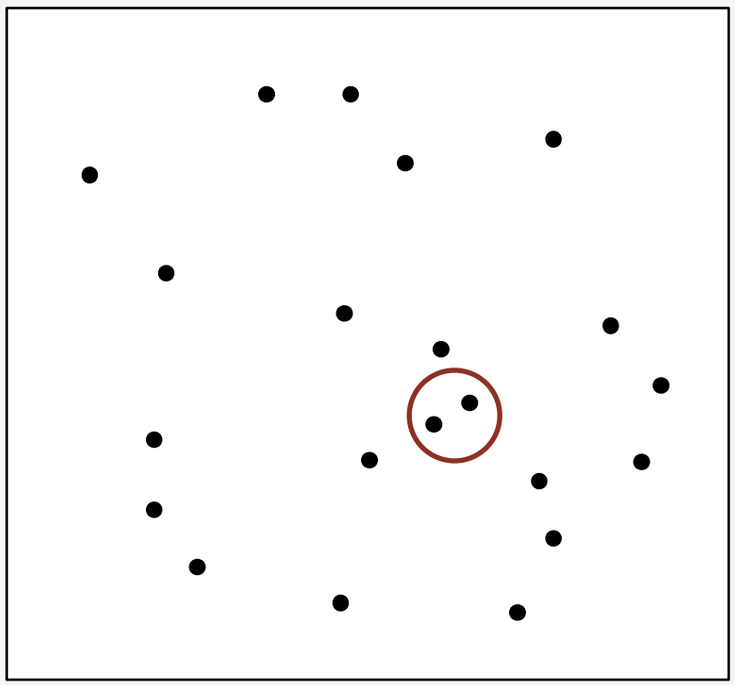
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



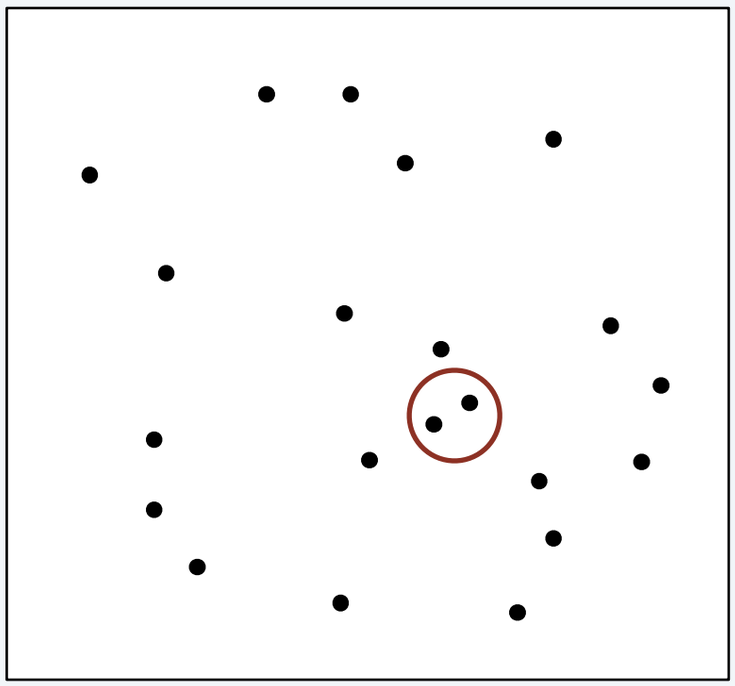
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

1D version. Easy $O(n \log n)$ algorithm if points are on a line.

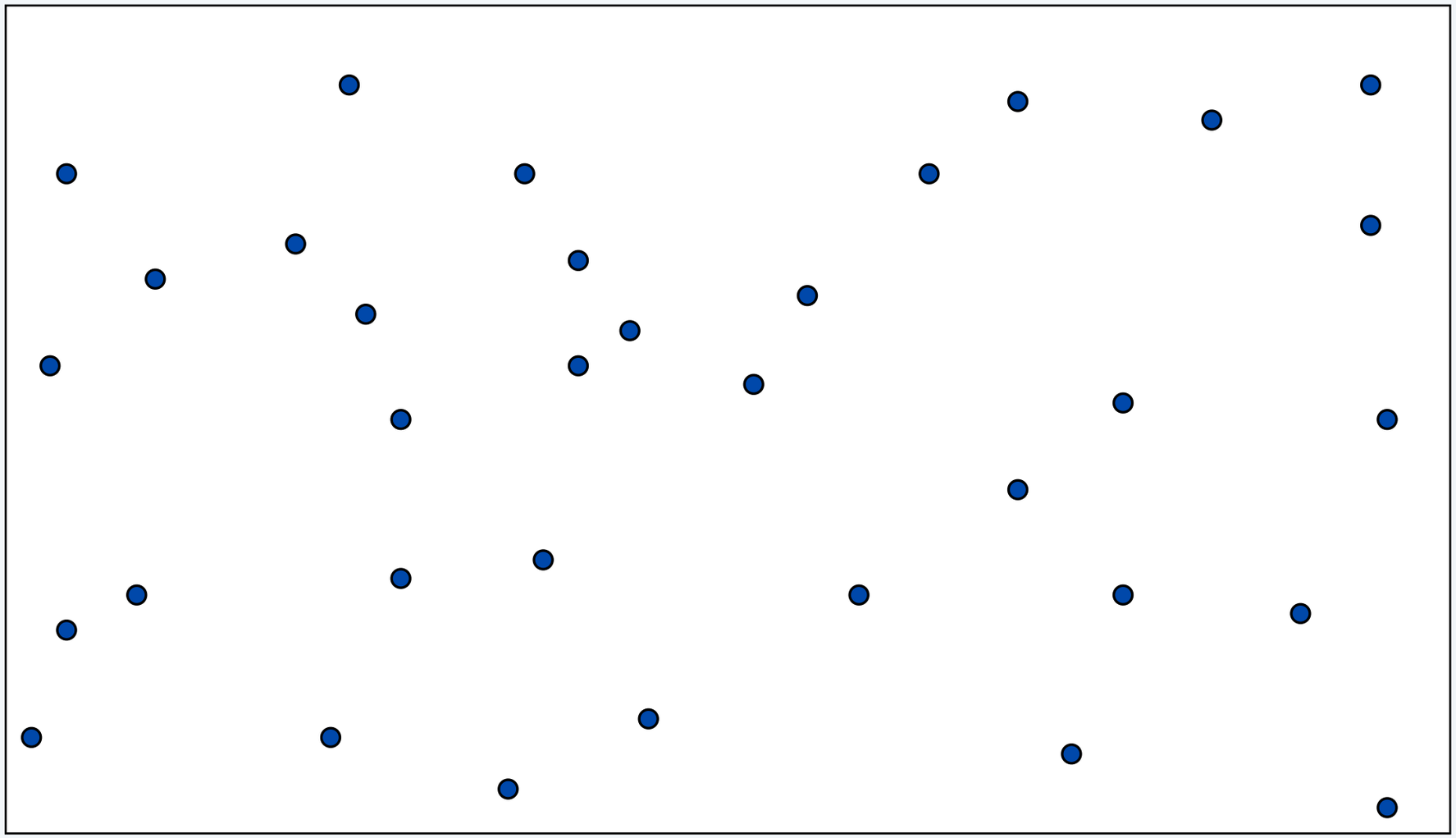
Non-degeneracy assumption. No two points have the same x-coordinate.



Closest pair of points: first attempt

Sorting solution.

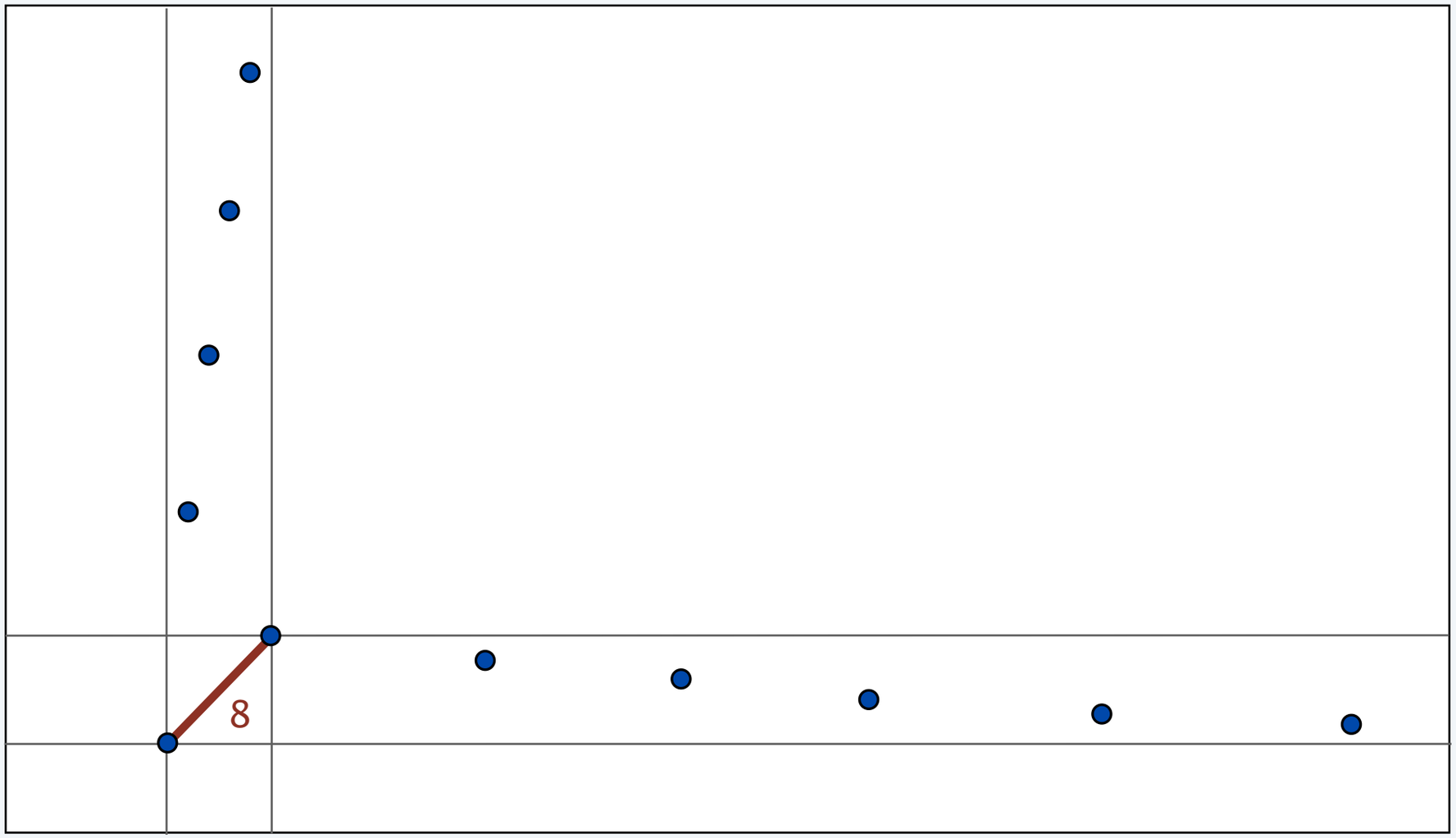
- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest pair of points: first attempt

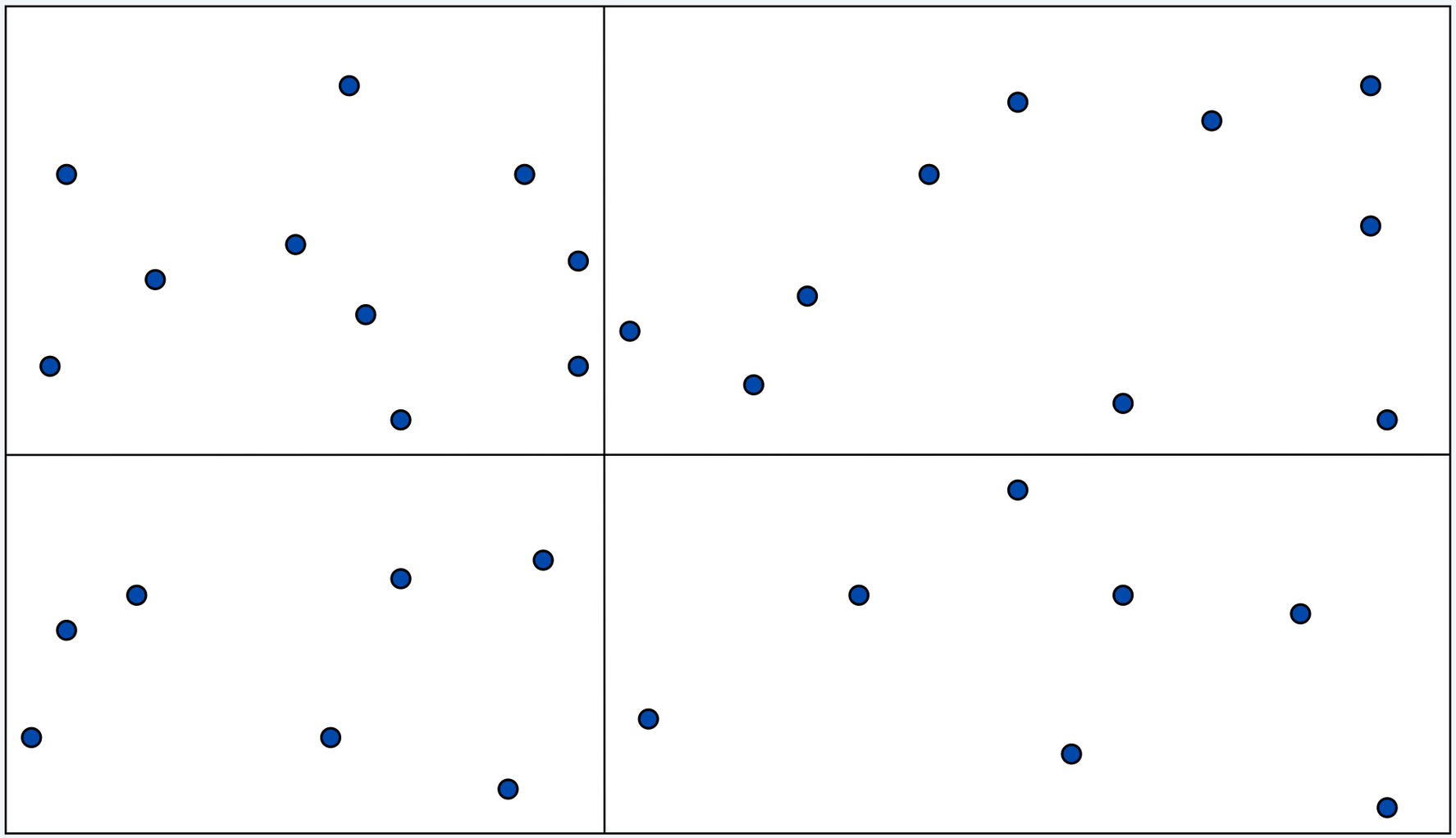
Sorting solution.

- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest pair of points: second attempt

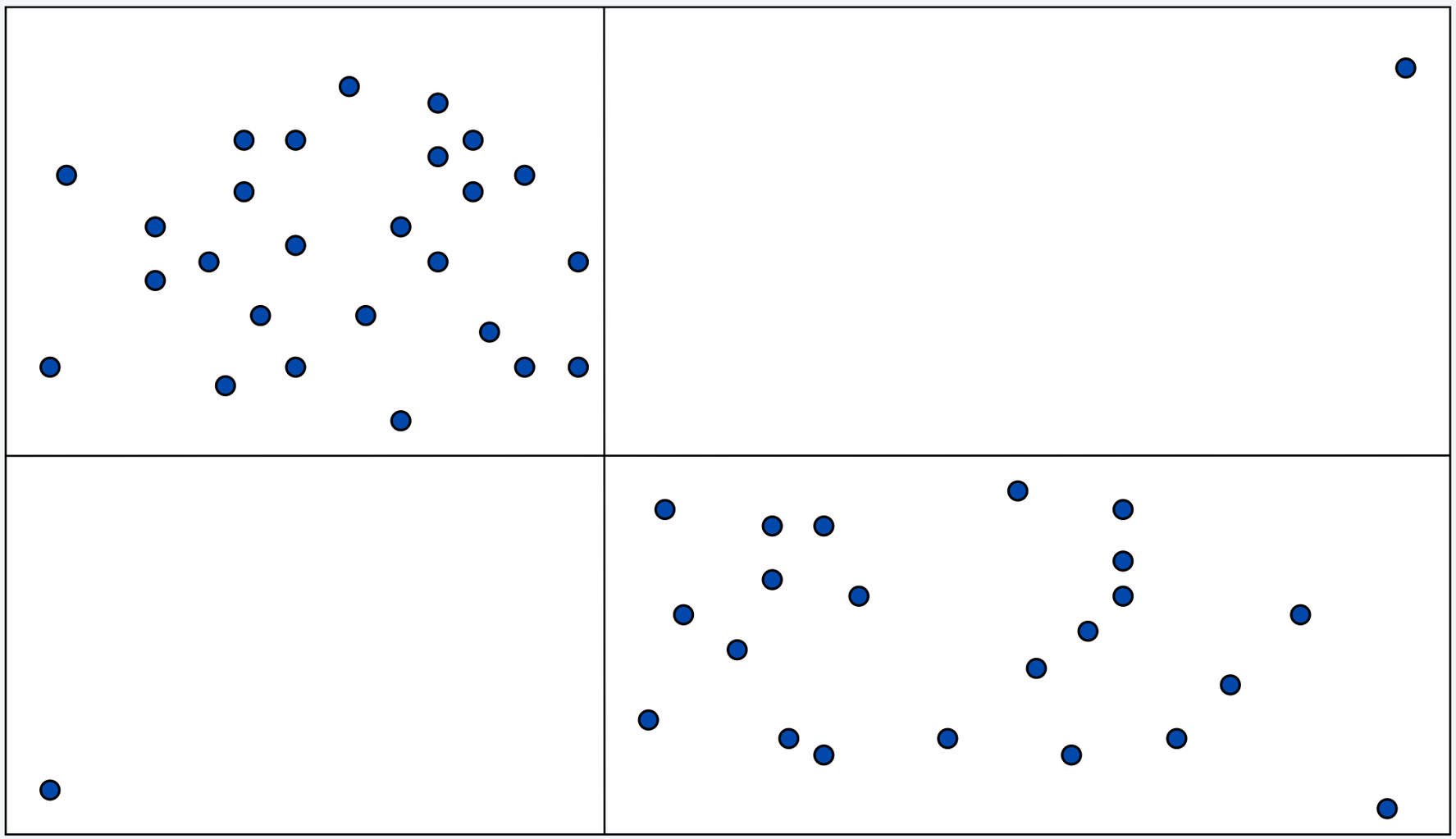
Divide. Subdivide region into 4 quadrants.



Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

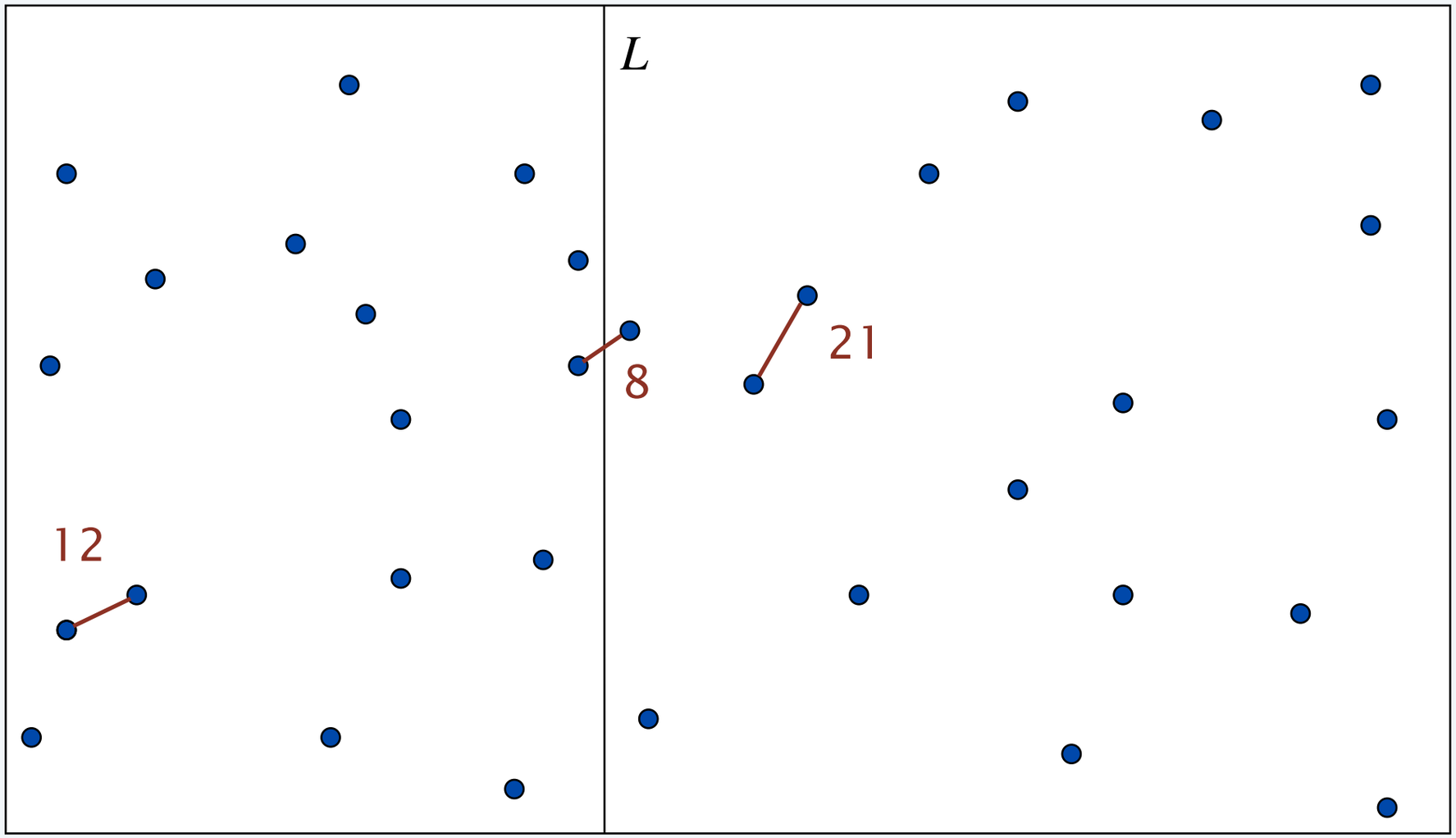
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

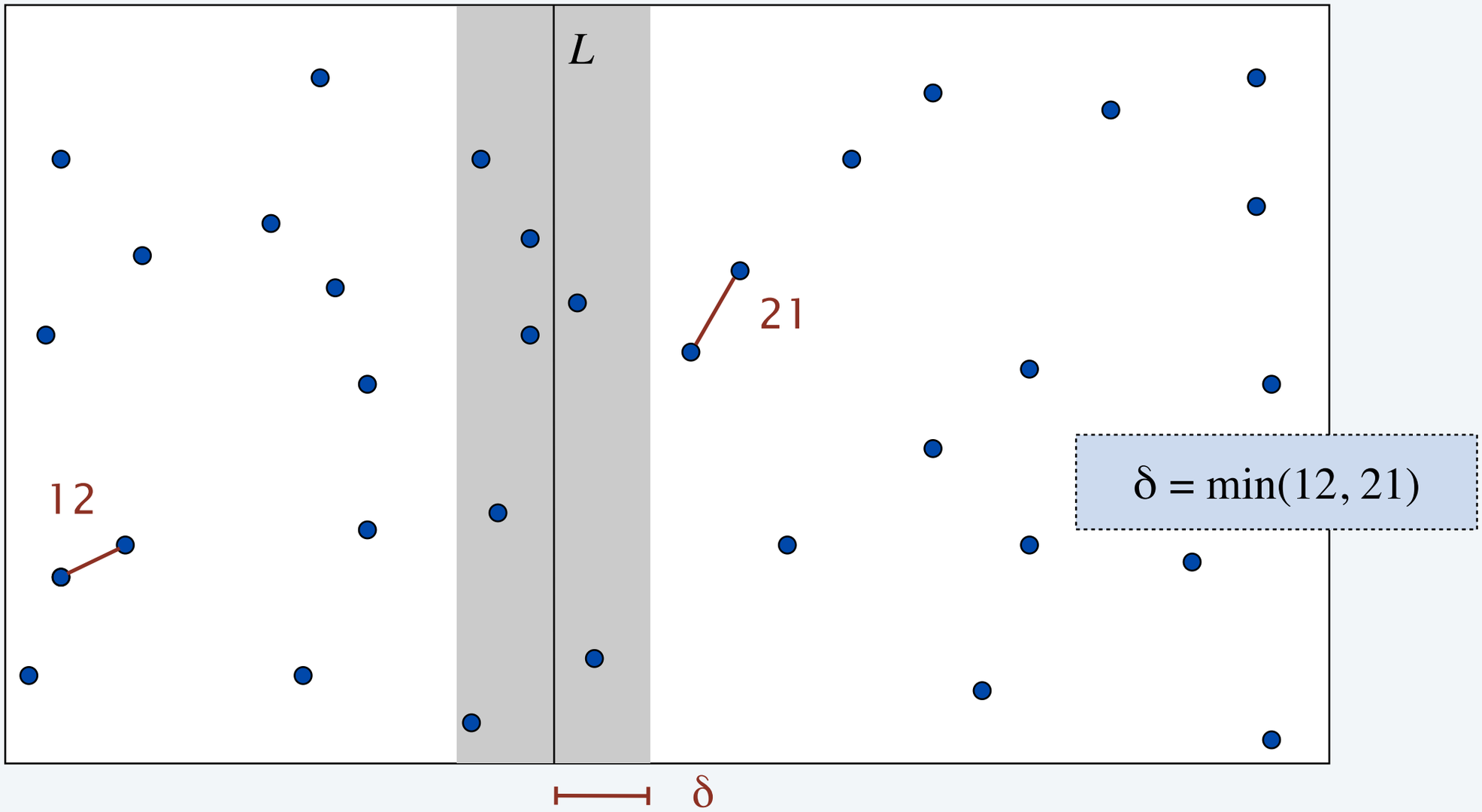
seems like $\Theta(n^2)$



How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: suffices to consider only those points within δ of line L.

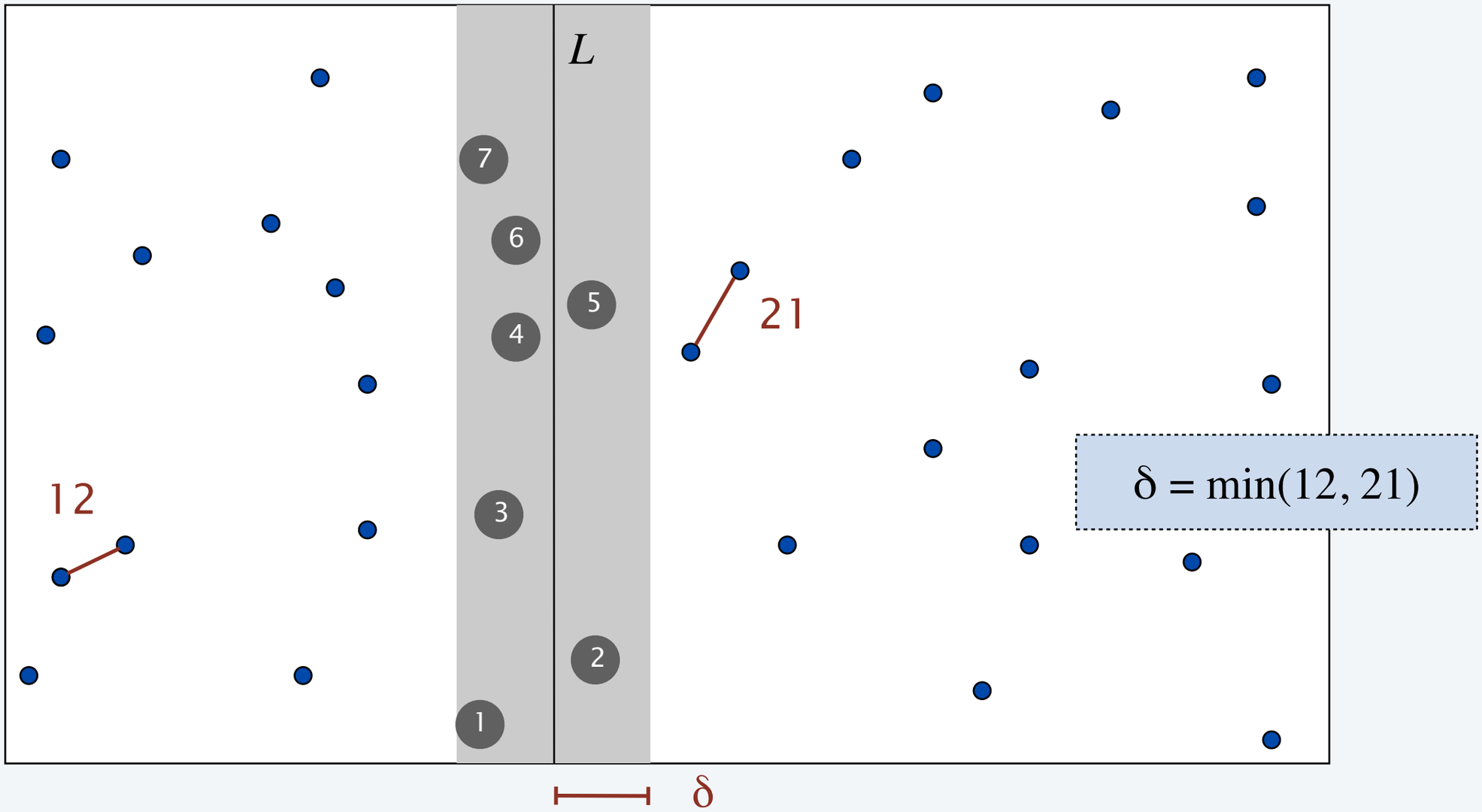


How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: suffices to consider only those points within δ of line L.
- Sort points in 2δ-strip by their y-coordinate.
- Check distances of only those points within 7 positions in sorted list!

why?



How to find closest pair with one point in each side?

Def. Let s_i be the point in the $2\,\delta$-strip, with the i^{th} smallest y-coordinate.

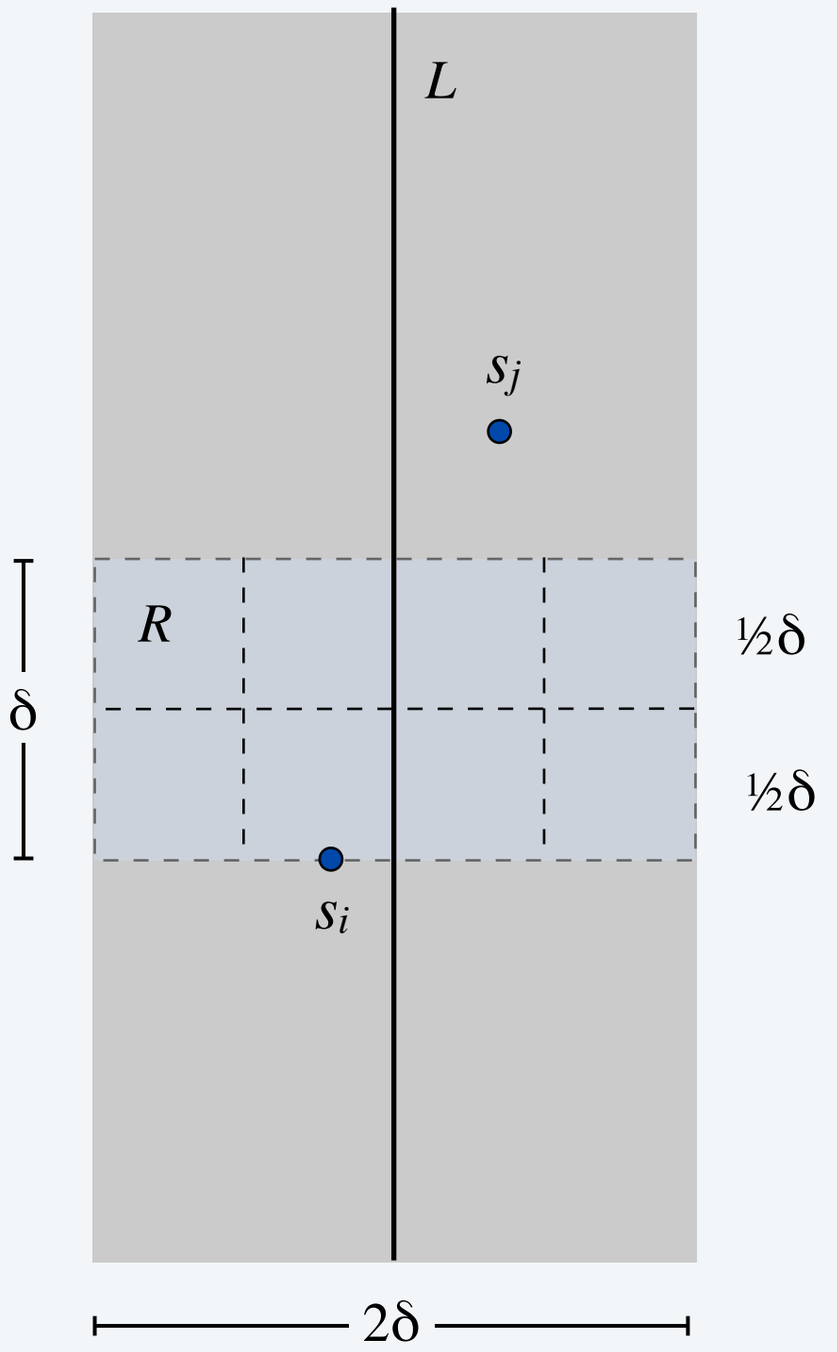
Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ.

Pf.

- Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
- Distance between s_i and any point s_j above R is $\geq \delta$.
- Subdivide R into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in R. ▪

diameter is $\delta / \sqrt{2} < \delta$

constant can be improved with more refined geometric packing argument



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR($p_1, p_2, \ldots, p_n$)

Compute vertical line L such that half the points are on each side of the line. ← $O(n)$

$\delta_1 \leftarrow$ CLOSEST-PAIR(points in left half). ← $T(n / 2)$

$\delta_2 \leftarrow$ CLOSEST-PAIR(points in right half). ← $T(n / 2)$

$\delta \leftarrow \min \{ \delta_1 , \delta_2 \}$.

Delete all points further than δ from line L. ← $O(n)$

Sort remaining points by y-coordinate. ← $O(n \log n)$

Scan points in y-order and compare distance between each point and next 7 neighbors. If any of these distances is less than δ, update δ. ← $O(n)$

RETURN δ.

Divide-and-conquer: quiz 6

What is the solution to the following recurrence?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n) & \text{if } n > 1 \end{cases}$$

A. $T(n) = \Theta(n)$.

B. $T(n) = \Theta(n \log n)$.

C. $T(n) = \Theta(n \log^2 n)$.

D. $T(n) = \Theta(n^2)$.

Refined version of closest-pair algorithm

Q. How to improve to $O(n \log n)$?

A. Don't sort points in strip from scratch each time.

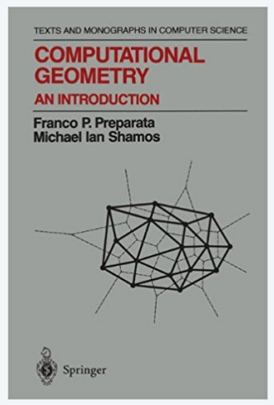
- Each recursive call returns two lists: all points sorted by x-coordinate, and all points sorted by y-coordinate.
- Sort by merging two pre-sorted lists.

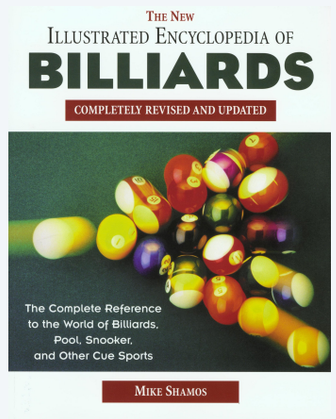
Either pre-sort beforehand
Or do merge sort along with recurrence
Check our exercise

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding a closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$





Divide-and-conquer: quiz 7

What is the complexity of the 2D closest pair problem?

A. $\Theta(n)$.

B. $\Theta(n \log^* n)$.

C. $\Theta(n \log \log n)$.

D. $\Theta(n \log n)$.

E. Not even Tarjan knows.

Computational complexity of closest-pair problem

Theorem. [Ben-Or 1983, Yao 1989] In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.

$(x_1 - x_2)^2 + (y_1 - y_2)^2$

Lower Bounds for Algebraic Computation Trees

with Integer Inputs*

Andrew Chi-Chih Yao
Department of Computer Science
Princeton University
Princeton, New Jersey 08544

Theorem. [Rabin 1976] There exists an algorithm to find the closest pair of points in the plane whose expected running time is $O(n)$.

not subject to $\Omega(n \log n)$ lower bound because it uses the floor function

A NOTE ON RABIN'S NEAREST-NEIGHBOR ALGORITHM*

Steve FORTUNE and John HOPCROFT
Department of Computer Science, Cornell University, Ithaca, NY, U.S.A.

Received 20 July 1978, revised version received 21 August 1978

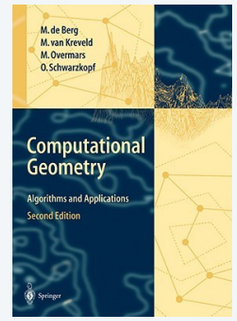
Probabilistic algorithms, nearest neighbor, hashing

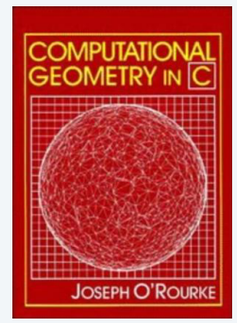
Digression: computational geometry

Ingenious divide-and-conquer algorithms for core geometric problems.

problem	brute	clever
closest pair	$O(n^2)$	$O(n \log n)$
farthest pair	$O(n^2)$	$O(n \log n)$
convex hull	$O(n^2)$	$O(n \log n)$
Delaunay/Voronoi	$O(n^4)$	$O(n \log n)$
Euclidean MST	$O(n^2)$	$O(n \log n)$

running time to solve a 2D problem with n points

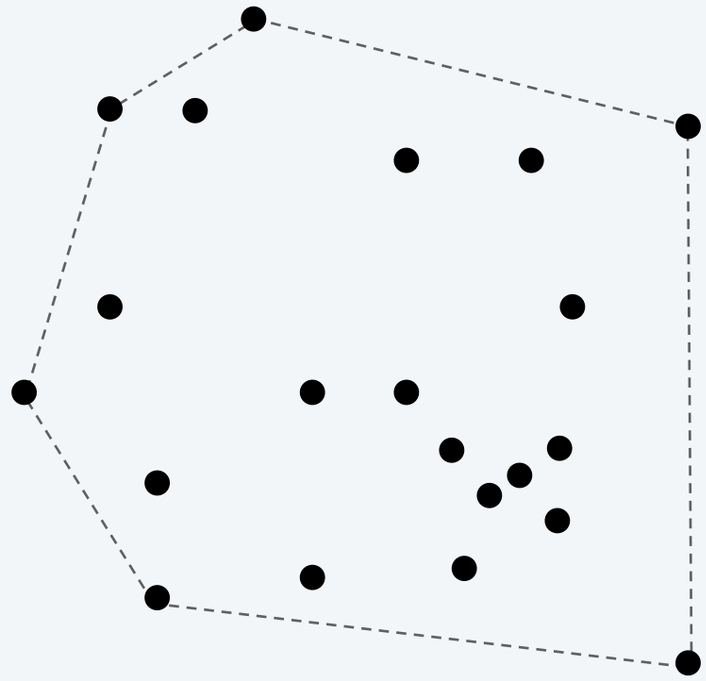




Note. 3D and higher dimensions test limits of our ingenuity.

Convex hull

The convex hull of a set of n points is the smallest perimeter fence enclosing the points.

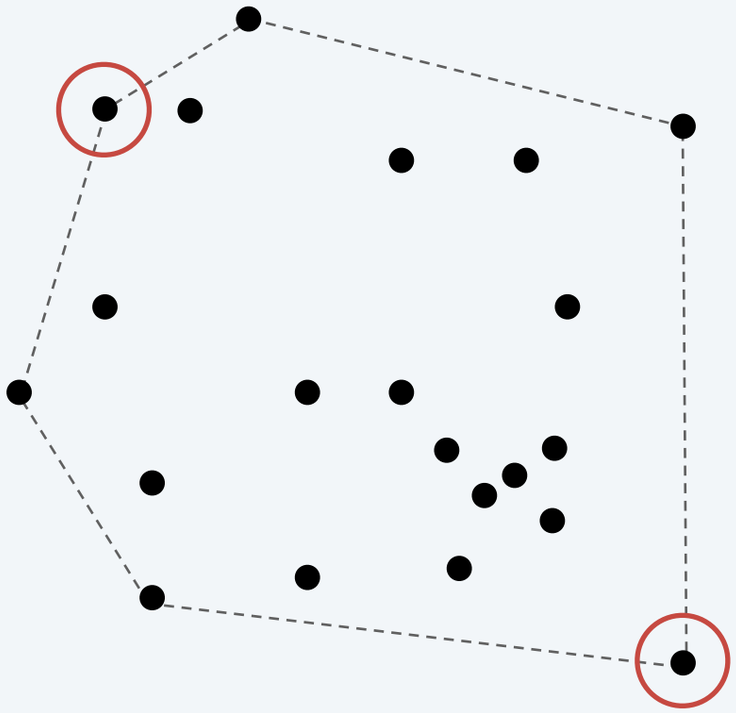


Equivalent definitions.

- Smallest area convex polygon enclosing the points.
- Intersection of all convex set containing all the points.

Farthest pair

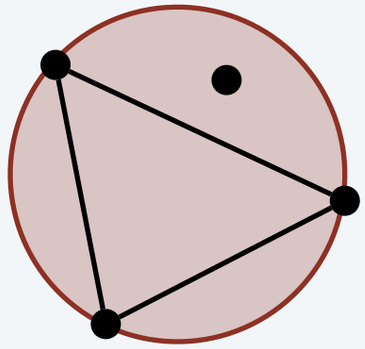
Given n points in the plane, find a pair of points with the largest Euclidean distance between them.



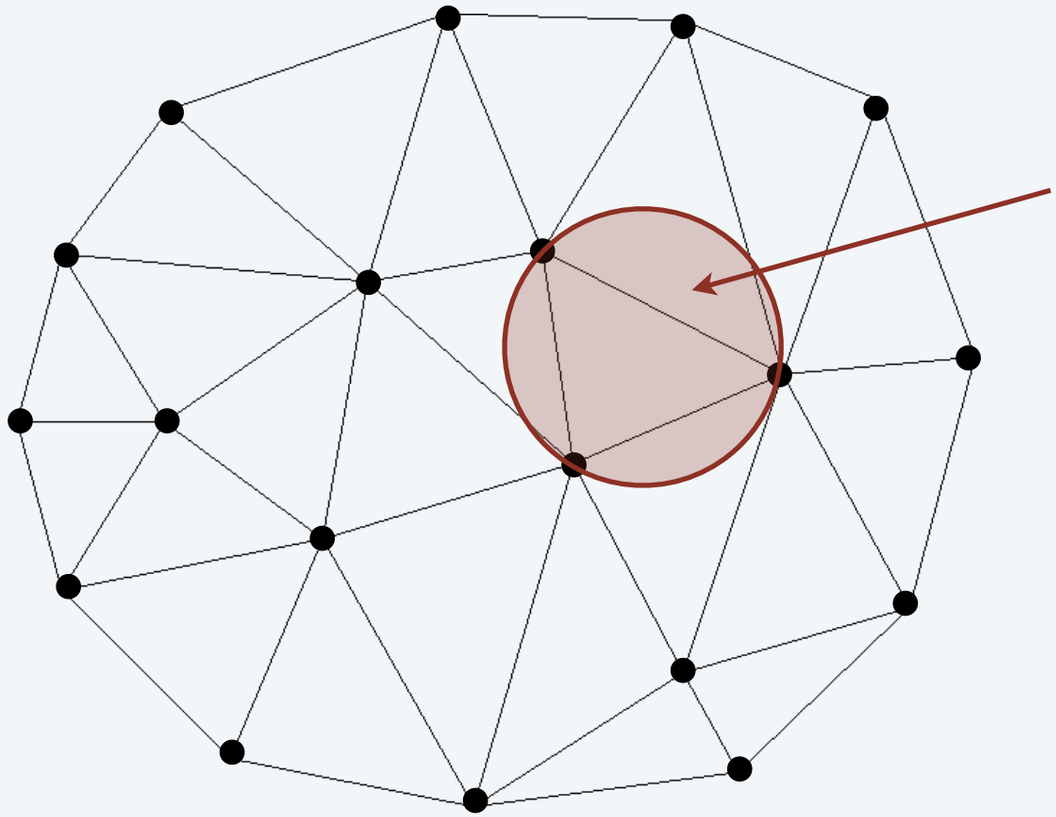
Fact. Points in farthest pair are extreme points on convex hull.

Delaunay triangulation

The Delaunay triangulation is a triangulation of n points in the plane such that no point is inside the circumcircle of any triangle.



point inside circumcircle of 3 points



Delaunay triangulation of 19 points

Some useful properties.

- No edges cross.
- Among all triangulations, it maximizes the minimum angle.
- Contains an edge between each point and its nearest neighbor.

Euclidean MST

Given n points in the plane, find MST connecting them.

[distances between point pairs are Euclidean distances]

Fact. Euclidean MST is subgraph of Delaunay triangulation.

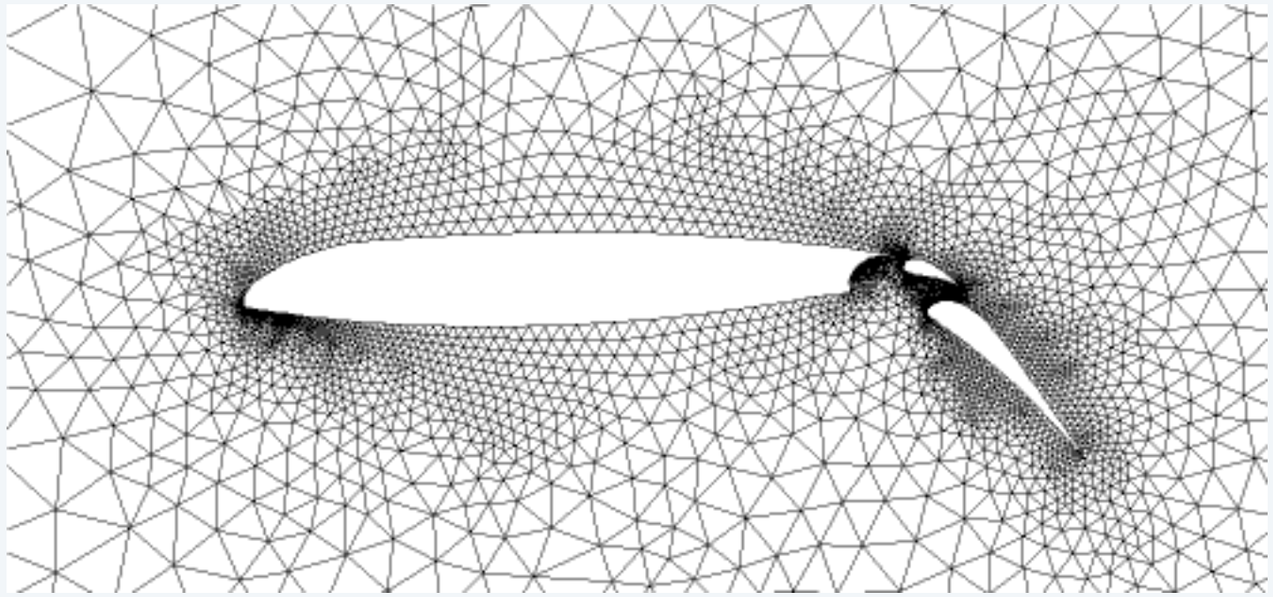
Implication. Can compute Euclidean MST in $O(n \log n)$ time.

- Compute Delaunay triangulation.
- Compute MST of Delaunay triangulation. ← it's planar ($\leq 3n$ edges)

Computational geometry applications

Applications.

- Robotics.
- VLSI design.
- Data mining.
- Medical imaging.
- Computer vision.
- Scientific computing.
- Finite-element meshing.
- Astronomical simulation.
- Models of physical world.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).



airflow around an aircraft wing

http://www.ics.uci.edu/~eppstein/geom.html

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
- Take-home messages

Three methods for solving recurrences:

$$T(N) = a\,T(N/b) + f(N)$$

- ☞ **Substitution method**
- ☞ **Recursion-tree method**
- ☞ **Master method**

✂ **Details to be ignored:**

- 👌 **if (N / b) is an integer or not**
- 👌 **always assume $T(n) = \Theta(1)$ for small n**

☞ **Substitution method — guess, then prove by induction**

〖**Example**〗 $T(N) = 2\,T(\lfloor N/2 \rfloor) + N$

Guess: $T(N) = O(N \log N)$

Proof: **Assume it is true for all** $m < N$, **in particular for** $m = \lfloor N/2 \rfloor$.

Then there exists a constant $c > 0$ **so that** $T(\lfloor N/2 \rfloor) \le c \lfloor N/2 \rfloor \log \lfloor N/2 \rfloor$

Substituting into the re

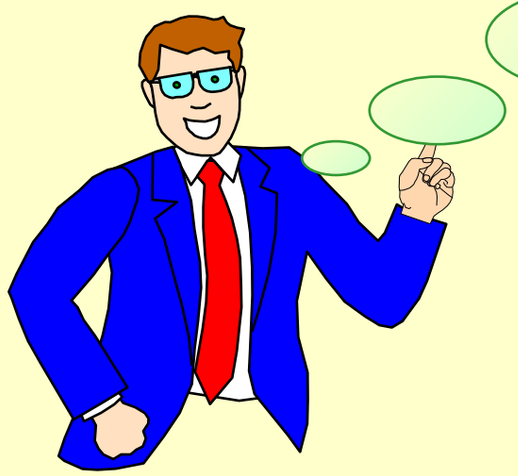
$T(N) = 2\,T$

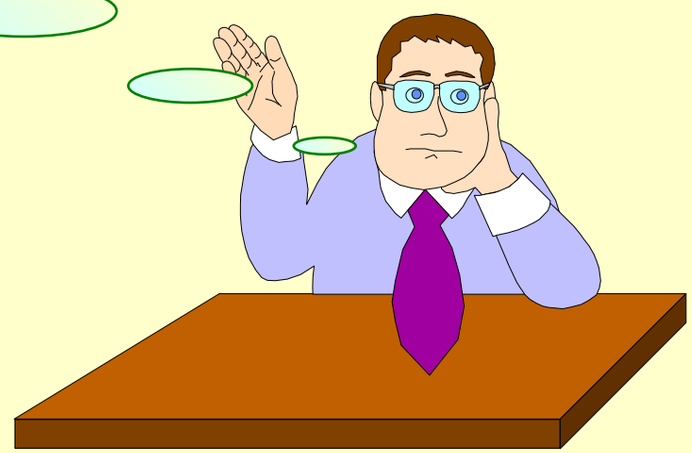
$\le$

$\le$

$\le c$

Relax!
As long as we can choose sufficiently large c **so that it is true for** $T(2)$ **and** $T(3)$.





〖Example〗 $T(N) = 2\,T(\lfloor N/2 \rfloor) + N$

Wrong guess: $T(N) = O(N)$

Proof: Assume it is true for all $m < N$, in particular for $m = \lfloor N/2 \rfloor$.

$$T(\lfloor N/2 \rfloor) \le c\,\lfloor N/2 \rfloor$$

Substituting into the recurrence:

$$T(N) = 2\,T(\lfloor N/2 \rfloor) + N$$
$$\le 2\,c\,\lfloor N/2 \rfloor + N$$
$$\le cN + N = O(N)$$ ✗

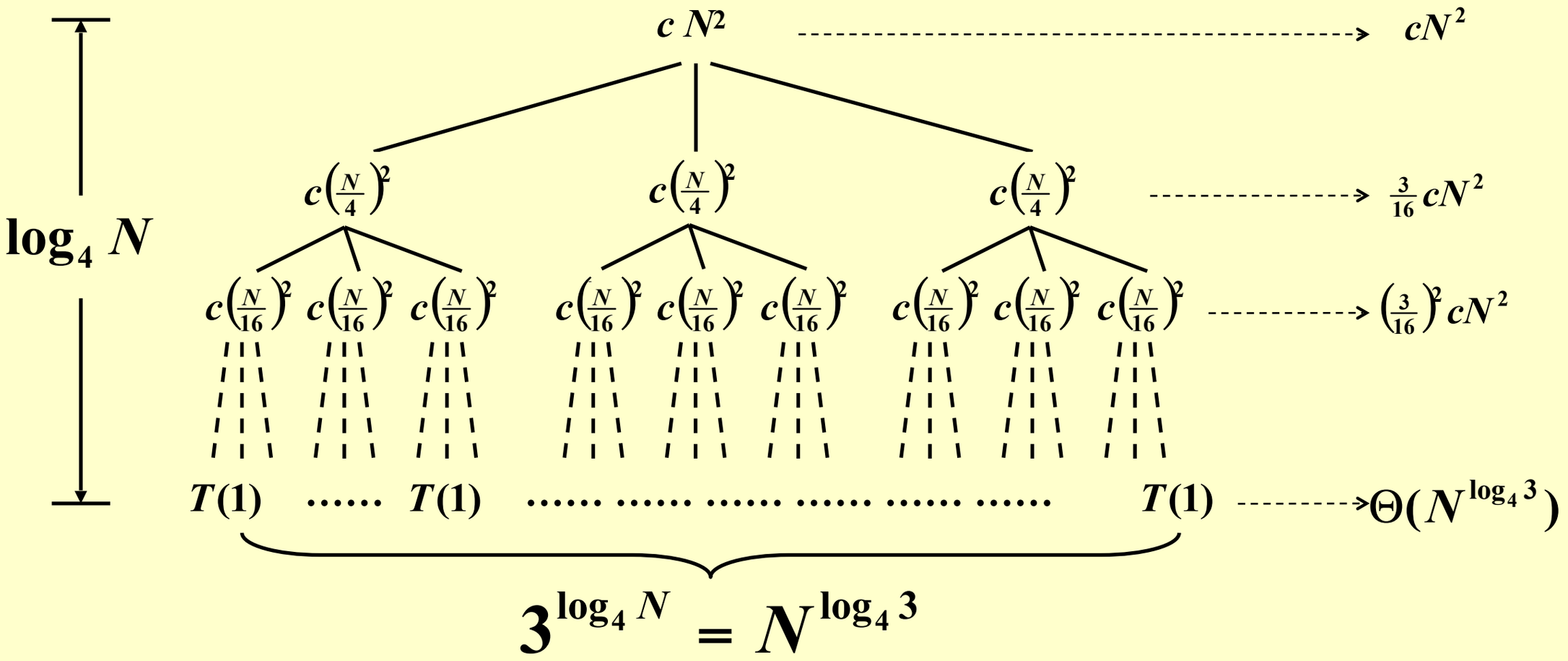
If we use O(N) here, then we cannot guarantee that there is a fixed constant c holds for all N

Must prove the *exact form*

How to make a good guess?

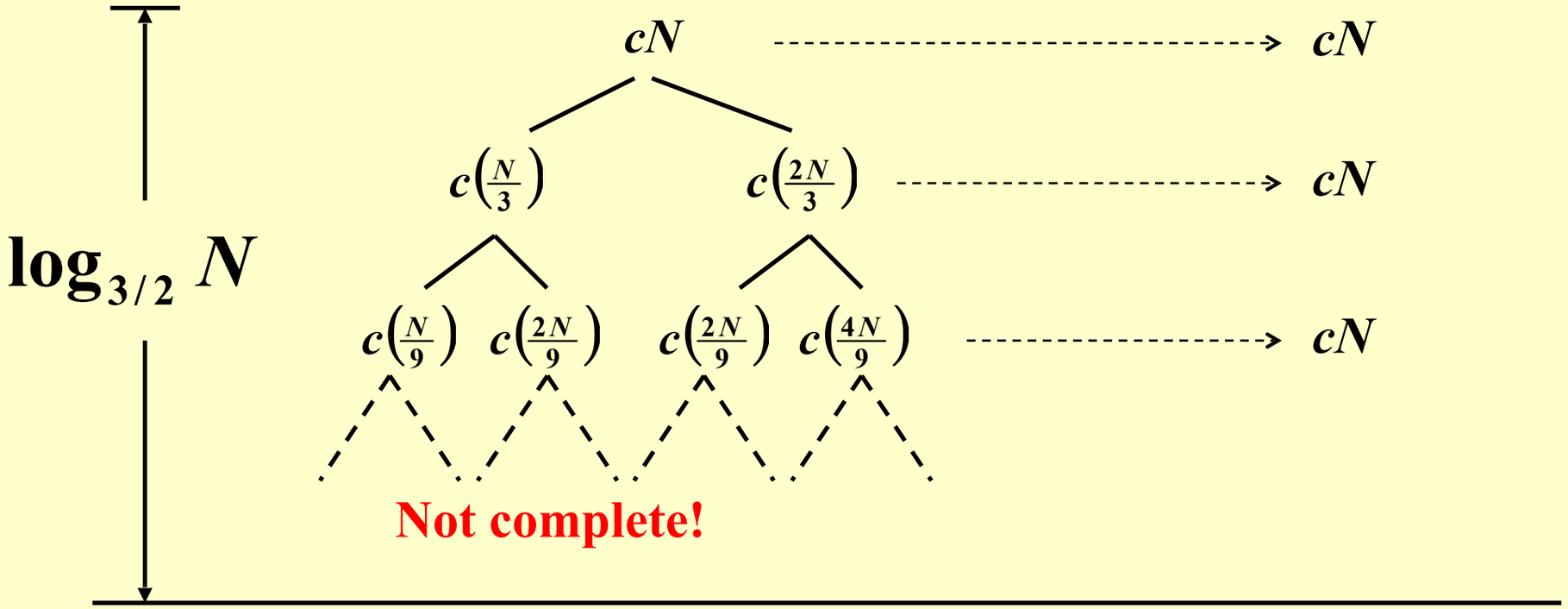
☞ **Recursion-tree method**

〖Example〗 $T(N) = 3\,T(N/4) + \Theta(N^2)$



$$T(N) = \sum_{i=0}^{\log_4 N - 1} \left(\frac{3}{16}\right)^i cN^2 + \Theta(N^{\log_4 3})$$

〖Example〗 $T(N) = T(N/3) + T(2N/3) + cN$



Guess: $O(N \log N)$

Proof by substitution:

$$T(N) = T(N/3) + T(2N/3) + cN \le d(N/3)\log(N/3) + d(2N/3)\log(2N/3) + cN$$

$$= dN \log N - dN(\log_2 3 - \tfrac{2}{3}) + cN \le dN \log N$$

for $d \ge c/(\log_2 3 - \tfrac{2}{3})$

Divide-and-conquer recurrences

Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

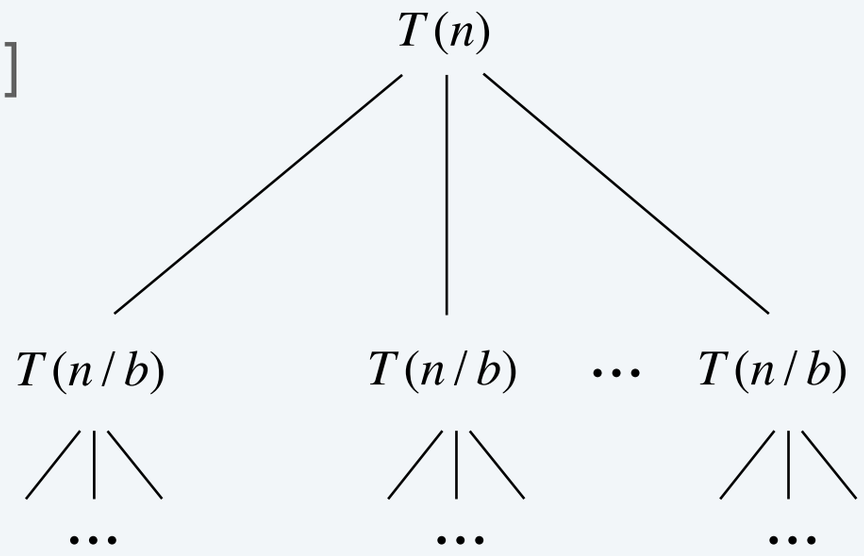
with $T(0) = 0$ and $T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.

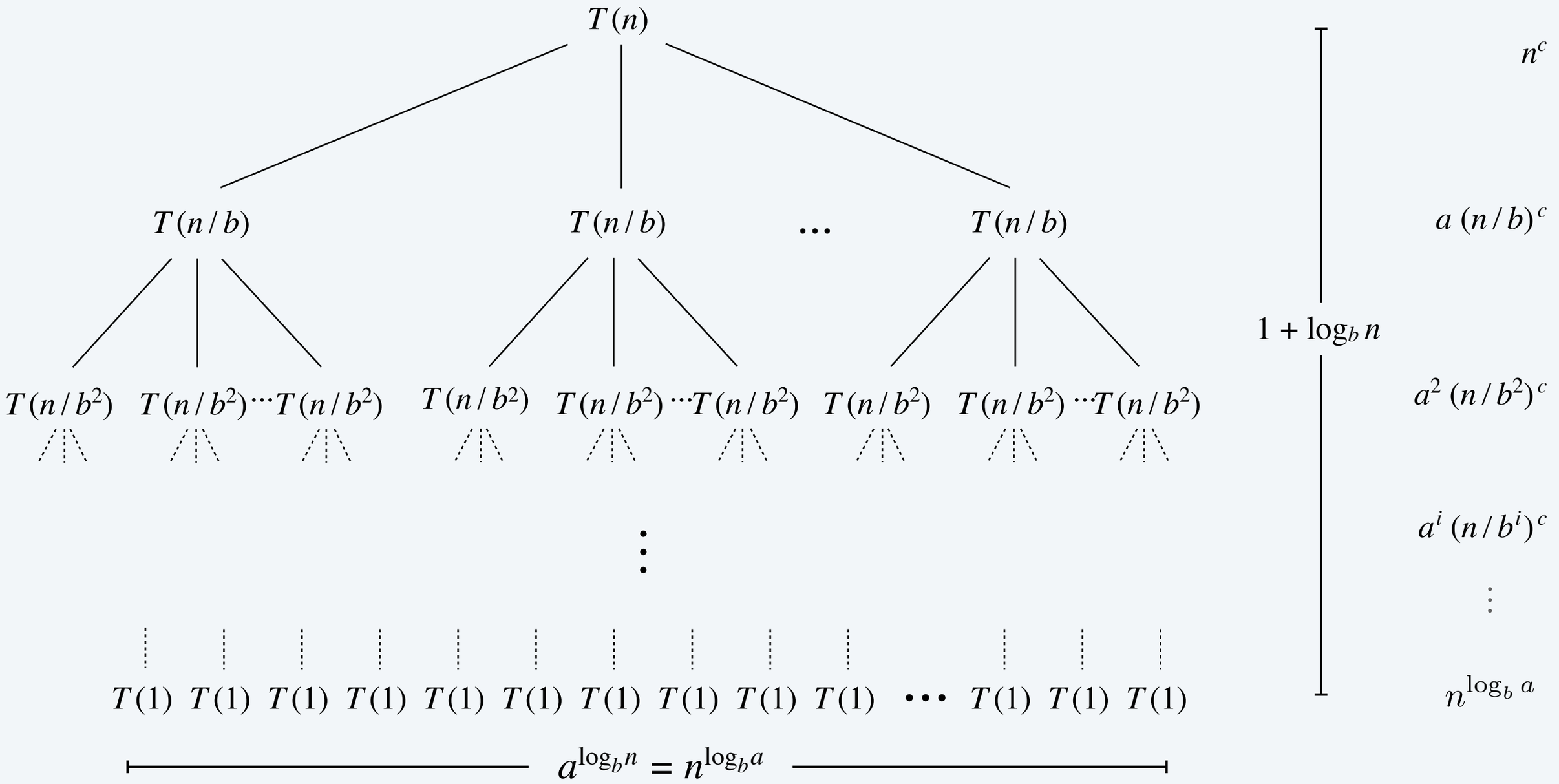
Recursion tree. [assuming n is a power of b]

- a = branching factor.
- a^i = number of subproblems at level i.
- $1 + \log_b n$ levels.
- n / b^i = size of subproblem at level i.



Divide-and-conquer recurrences: recursion tree

Suppose $T(n)$ satisfies $T(n) = a\,T(n/b) + n^c$ with $T(1) = 1$, for n a power of b.



$r = a / b^c$ $\qquad T(n) = n^c \sum_{i=0}^{\log_b n} r^i$

Divide-and-conquer recurrences: recursion tree analysis

Suppose $T(n)$ satisfies $T(n) = a\,T(n/b) + n^c$ with $T(1) = 1$, for n a power of b.

Let $r = a / b^c$. Note that $r < 1$ iff $c > \log_b a$.

$$T(n) \;=\; n^c \sum_{i=0}^{\log_b n} r^i \;=\; \begin{cases} \Theta(n^c) & \text{if } r < 1 \\ \Theta(n^c \log n) & \text{if } r = 1 \\ \Theta(n^{\log_b a}) & \text{if } r > 1 \end{cases}$$

$c > \log_b a$ ← cost dominated by cost of root

$c = \log_b a$ ← cost evenly distributed in tree

$c < \log_b a$ ← cost dominated by cost of leaves

Geometric series.

- If $0 < r < 1$, then $1 + r + r^2 + r^3 + \ldots + r^k \;\le\; 1 / (1 - r)$.
- If $r = 1$, then $1 + r + r^2 + r^3 + \ldots + r^k \;=\; k + 1$.
- If $r > 1$, then $1 + r + r^2 + r^3 + \ldots + r^k \;= (r^{k+1} - 1) \;/ (r - 1)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.


D. Blostein Jon Bentley

James B. Saxe

Pf sketch.

- Prove when b is an integer and n is an exact power of b.
- Extend domain of recurrences to reals (or rationals).
- Deal with floors and ceilings. ⟵ at most 2 extra levels in recursion tree

$$\begin{aligned} \lceil \lceil \lceil n/b \rceil / b \rceil / b \rceil \rceil &< n/b^3 + (1/b^2 + 1/b + 1) \\ &\leq n/b^3 + 2 \end{aligned}$$

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Extensions.

- Can replace Θ with O everywhere.
- Can replace Θ with Ω everywhere.
- Can replace initial conditions with $T(n) = \Theta(1)$ for all $n \leq n_0$ and require recurrence to hold only for all $n > n_0$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 1] $T(n) = 3\,T(\lfloor n/2 \rfloor) + 5n$.

- $a = 3,\ b = 2,\ c = 1 < \log_b a = 1.5849\ldots$.
- $T(n) = \Theta(n^{\log_2 3}) = \mathrm{O}(n^{1.58})$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

ok to intermix floor and ceiling

Ex. [Case 2] $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 17n$.

- $a = 2,\ b = 2, c = 1 = \log_b a$.
- $T(n) = \Theta(n \log n)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 3] $T(n) = 48\,T(\lfloor n/4 \rfloor) + n^3$.

- $a = 48,\ b = 4,\ c = 3 > \log_b a = 2.7924....$
- $T(n) = \Theta(n^3)$.

Master theorem need not apply

Gaps in master theorem.

- Number of subproblems is not a constant.

$$T(n) = n\,T(n/2) + n^2$$

- Number of subproblems is less than 1.

$$T(n) = \frac{1}{2}\,T(n/2) + n^2$$

- Work to divide and combine subproblems is not $\Theta(n^c)$.

$$T(n) = 2\,T(n/2) + n \log n$$

Note: This is actually not limited.
The master theorem has a more general version to handle logarithm in f(n).

Theorem 4.1 (Master theorem)
Let $a > 0$ and $b > 1$ be constants, and let $f(n)$ be a driving function that is defined and nonnegative on all sufficiently large reals. Define the recurrence $T(n)$ on $n \in \mathbb{N}$ by

$$T(n) = aT(n/b) + f(n) , \tag{4.17}$$

where $aT(n/b)$ actually means $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ for some constants $a' \geq 0$ and $a'' \geq 0$ satisfying $a = a' + a''$. Then the asymptotic behavior of $T(n)$ can be characterized as follows:

1. If there exists a constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.
2. If there exists a constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
3. If there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and if $f(n)$ additionally satisfies the ***regularity condition*** $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

From CLRS 4th edition. Sec. 4.5.
The last theorem in our official slides is actual similar, easier to use, but less rigorous (For our course, may be better to use it).

Akra–Bazzi theorem

Examples of functions that satisfy the polynomial-growth condition include any function of the form $f(n) = \Theta(n^\alpha \lg^\beta n \lg\lg^\gamma n)$, where α, β, and γ are constants.

CLRS Sec. 4.7

Theorem. [Akra–Bazzi 1998] Given constants $a_i > 0$ and $0 < b_i < 1$, functions $|h_i(n)| = O(n / \log^2 n)$ and $g(n) = O(n^c)$. If $T(n)$ satisfies the recurrence:

Also not limited to this form

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + g(n)$$

a_i subproblems of size $b_i\, n$

small perturbation to handle floors and ceilings

then, $T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right)$, where p satisfies $\sum_{i=1}^{k} a_i\, b_i^p = 1$.

Ex. $T(n) = T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5\, n$, with $T(0) = 0$ and $T(1) = 0$.

- $a_1 = 1,\ b_1 = 1/5,\ a_2 = 1,\ b_2 = 7/10 \Rightarrow p = 0.83978\ldots < 1.$
- $h_1(n) = \lfloor n/5 \rfloor - n/5,\ h_2(n) = 3/10\, n - 3\lfloor n/10 \rfloor.$
- $g(n) = 11/5\, n \Rightarrow T(n) = \Theta(n).$

☞ **Master method**

【Master Theorem】Let $a \geq 1$ and $b > 1$ be constants, let $f(N)$ be a function, and let $T(N)$ be defined on the nonnegative integers by the recurrence $T(N) = aT(N/b) + f(N)$. Then:

1. If $f(N) = O(N^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(N) = \Theta(N^{\log_b a})$
2. If $f(N) = \Theta(N^{\log_b a})$, then $T(N) = \Theta(N^{\log_b a} \log N)$
3. If $f(N) = \Omega(N^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(N/b) < cf(N)$ for some constant $c < 1$ and all sufficiently large N, then $T(N) = \Theta(f(N))$

regularity condition

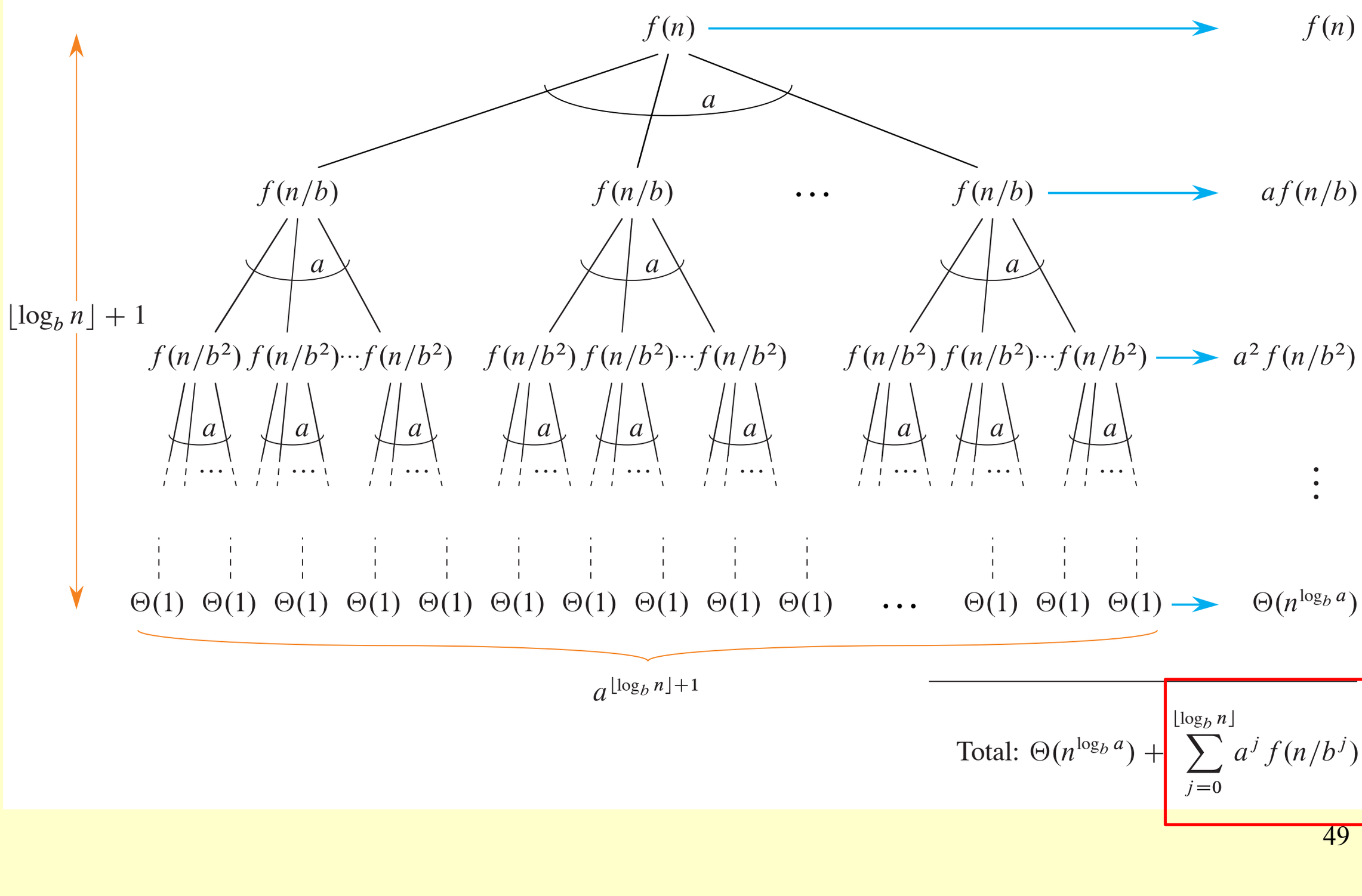
〖Example〗 Mergesort has $a = b = 2$, and case 2

→ $T = \mathrm{O}(N \log N)$

〖Example〗 $a = b = 2$, $f(N) = N\log N$?

→ $T = \mathrm{O}(N \log N)$

Proof by recursion tree: for $n = b^k$ for some integer k



For case 1 where $f(N) = O(N^{\log_b a - \varepsilon})$

$$\sum_{j=0}^{\log_b N - 1} a^j f(N / b^j) =$$

$$= O(N^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b N - 1} (b^\varepsilon)^j) = O(N^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b N} - 1}{b^\varepsilon - 1})$$

$$= O(N^{\log_b a - \varepsilon} N^\varepsilon) = O(N^{\log_b a})$$

$$T(N) = \Theta(N^{\log_b a}) + O(N^{\log_b a}) = \Theta(N^{\log_b a})$$

Discussion 9:
Please prove case 2.

Read Ch.4 of "Introduction to Algorithms" for the rest of the proof.

☞ **Master method –** ***another form***

【Master Theorem】The recurrence $T(N) = aT(N/b) + f(N)$ **can be solved as follows:**

1. **If** $af(N/b) = \kappa f(N)$ **for some constant** $\kappa < 1$, **then** $T(N) = \Theta(f(N))$
2. **If** $af(N/b) = Kf(N)$ **for some constant** $K > 1$, **then** $T(N) = \Theta(N^{\log_b a})$
3. **If** $af(N/b) = f(N)$, **then** $T(N) = \Theta(f(N)\log_b N)$

〖Example〗 $a = 4, b = 2, f(N) = N\log N$

$$af(N/b) = 4(N/2)\log(N/2) = 2N\log N - 2N \quad ?$$

$$f(N) = N\log N \qquad O(N^{\log_b a - \varepsilon}) = O(N^{2-\varepsilon})$$

$\longrightarrow T = O(N^2)$

【Theorem】The solution to the equation

$$T(N) = a\,T(N/b) + \Theta(N^k \log^p N),$$

where $a \geq 1$, $b > 1$, and $p \geq 0$ is

$$T(N) = \begin{cases} O(N^{\log_b a}) & \text{if } a > b^k \\ O(N^k \log^{p+1} N) & \text{if } a = b^k \\ O(N^k \log^p N) & \text{if } a < b^k \end{cases}$$

〖Example〗 Mergesort has $a = b = 2$, $p = 0$ and $k = 1$.

→ $T = O(N \log N)$

〖Example〗 Divide with $a = 3$, and $b = 2$ for each recursion;
Conquer with $O(N)$ – that is, $k = 1$ and $p = 0$.

→ $T = O(N^{1.59})$

If conquer takes $O(N^2)$ then $T = O(N^2)$.

〖Example〗 $a = b = 2$, $f(N) = N\log N$ → $T = O(N \log^2 N)$

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
  - Integer multiplication
  - Matrix multiplication
  - Polynomial multiplication with FFT
- Take-home messages

Integer addition and subtraction

Addition. Given two n-bit integers a and b, compute $a + b$.

Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations. ← "bit complexity" (instead of word RAM)

	1	1	1	1	1	1	0	1
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

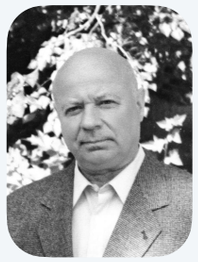
Remark. Grade-school addition and subtraction algorithms are optimal.

Integer multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm (long multiplication). $\Theta(n^2)$ bit operations.

	1	1	0	1	0	1	0	1
×	0	1	1	1	1	1	0	1
	1	1	0	1	0	1	0	1
0	0	0	0	0	0	0	0	
1	1	0	1	0	1	0	1	
1	1	0	1	0	1	0	1	
1	1	0	1	0	1	0	1	
1	1	0	1	0	1	0	1	
0	0	0	0	0	0	0	0	
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1



Conjecture. [Kolmogorov 1956] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is false.

Divide-and-conquer multiplication

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

← use bit shifting to compute 4 terms

$$x\,y = (2^m a + b)(2^m c + d) = \underset{1}{2^{2m} ac} + \underset{2}{2^m (bc} + \underset{3}{ad)} + \underset{4}{bd}$$

Ex. $x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b}$ $\quad y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$

Divide-and-conquer multiplication

MULTIPLY(x, y, n)

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

  $m \leftarrow \lceil n / 2 \rceil$.

  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

  $\Theta(n)$

  $e \leftarrow$ MULTIPLY(a, c, m).

  $f \leftarrow$ MULTIPLY(b, d, m).

  $g \leftarrow$ MULTIPLY(b, c, m).

  $h \leftarrow$ MULTIPLY(a, d, m).

  $4\,T(\lceil n / 2 \rceil)$

  RETURN $2^{2m}\, e + 2^m\, (g + h) + f$. $\Theta(n)$

Divide-and-conquer II: quiz 3

How many bit operations to multiply two n-bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

A. $T(n) = \Theta(n^{1/2})$.

B. $T(n) = \Theta(n \log n)$.

C. $T(n) = \Theta(n^{\log_2 3}) = \mathrm{O}(n^{1.585})$.

D. $T(n) = \Theta(n^2)$.

The mathematician Carl Friedrich Gauss (1777–1855) once noticed that although the product of two complex numbers

$$(a+bi)(c+di) \quad = \quad ac-bd+(bc+ad)i$$

seems to involve *four* real-number multiplications, it can in fact be done with just *three*: ac, bd, and $(a+b)(c+d)$, since

$$bc+ad \quad = \quad (a+b)(c+d)-ac-bd.$$

Karatsuba trick

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
- To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

- Multiply only three $\frac{1}{2}n$-bit integers, recursively.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

$$x = 1\underbrace{0\,0\,0}_{a}\underbrace{0\,1\,1\,0\,1}_{b}$$

$$y = 1\underbrace{1\,1\,0}_{c}\underbrace{0\,0\,0\,0\,1}_{d}$$

middle term

$$x\,y = (2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

1 1 3 2 3

Karatsuba multiplication

KARATSUBA-MULTIPLY(x, y, n)

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

  $m \leftarrow \lceil n / 2 \rceil$.

  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

  $\Theta(n)$

  $e \leftarrow$ KARATSUBA-MULTIPLY(a, c, m).

  $f \leftarrow$ KARATSUBA-MULTIPLY(b, d, m).

  $g \leftarrow$ KARATSUBA-MULTIPLY$(|a - b|, |c - d|, m)$.

  $3\ T(\lceil n / 2 \rceil)$

  Flip sign of g if needed.

  RETURN $2^{2m} e + 2^m (e + f - g) + f$. $\Theta(n)$

Karatsuba analysis

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two n-bit integers.

Pf. Apply Case 3 of the master theorem to the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$\Longrightarrow \quad T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

Practice.

- Use base 32 or 64 (instead of base 2).
- Faster than grade-school algorithm for about 320–640 bits.

Integer arithmetic reductions

Integer multiplication. Given two n-bit integers, compute their product.

arithmetic problem	formula	bit complexity
integer multiplication	$a \times b$	$M(n)$
integer square	a^2	$\Theta(M(n))$
integer division	$\lfloor a / b \rfloor,\ a \bmod b$	$\Theta(M(n))$
integer square root	$\lfloor \sqrt{a} \rfloor$	$\Theta(M(n))$

$$ab = \frac{(a+b)^2 - a^2 - b^2}{2}$$

integer arithmetic problems with the same bit complexity M(n) as integer multiplication

History of asymptotic complexity of integer multiplication

year	algorithm	bit operations
12xx	**grade school**	$O(n^2)$
1962	**Karatsuba–Ofman**	$O(n^{1.585})$
1963	**Toom-3, Toom-4**	$O(n^{1.465})$, $O(n^{1.404})$
1966	**Toom–Cook**	$O(n^{1+\varepsilon})$
1971	**Schönhage–Strassen**	$O(n \log n \cdot \log \log n)$
2007	**Fürer**	$n \log n \, 2^{O(\log^* n)}$
2019	**Harvey–van der Hoeven**	$O(n \log n)$
		$O(n)$

number of bit operations to multiply two n-bit integers

Remark. GNU Multiple Precision library uses one of first five algorithms depending on n.

used in Maple, Mathematica, gcc, cryptography, ...

GMP
«Arithmetic without limitations»

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
  - Integer multiplication
  - Matrix multiplication
  - Polynomial multiplication with FFT
- Take-home messages

Dot product

Dot product. Given two length-n vectors a and b, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^{n} a_i \, b_i$$

$$\begin{aligned} a &= \begin{bmatrix} .70 & .20 & .10 \end{bmatrix} \\ b &= \begin{bmatrix} .30 & .40 & .30 \end{bmatrix} \\ a \cdot b &= (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32 \end{aligned}$$

Remark. "Grade-school" dot product algorithm is asymptotically optimal.

Matrix multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute $\underbrace{C = AB}_{n}$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

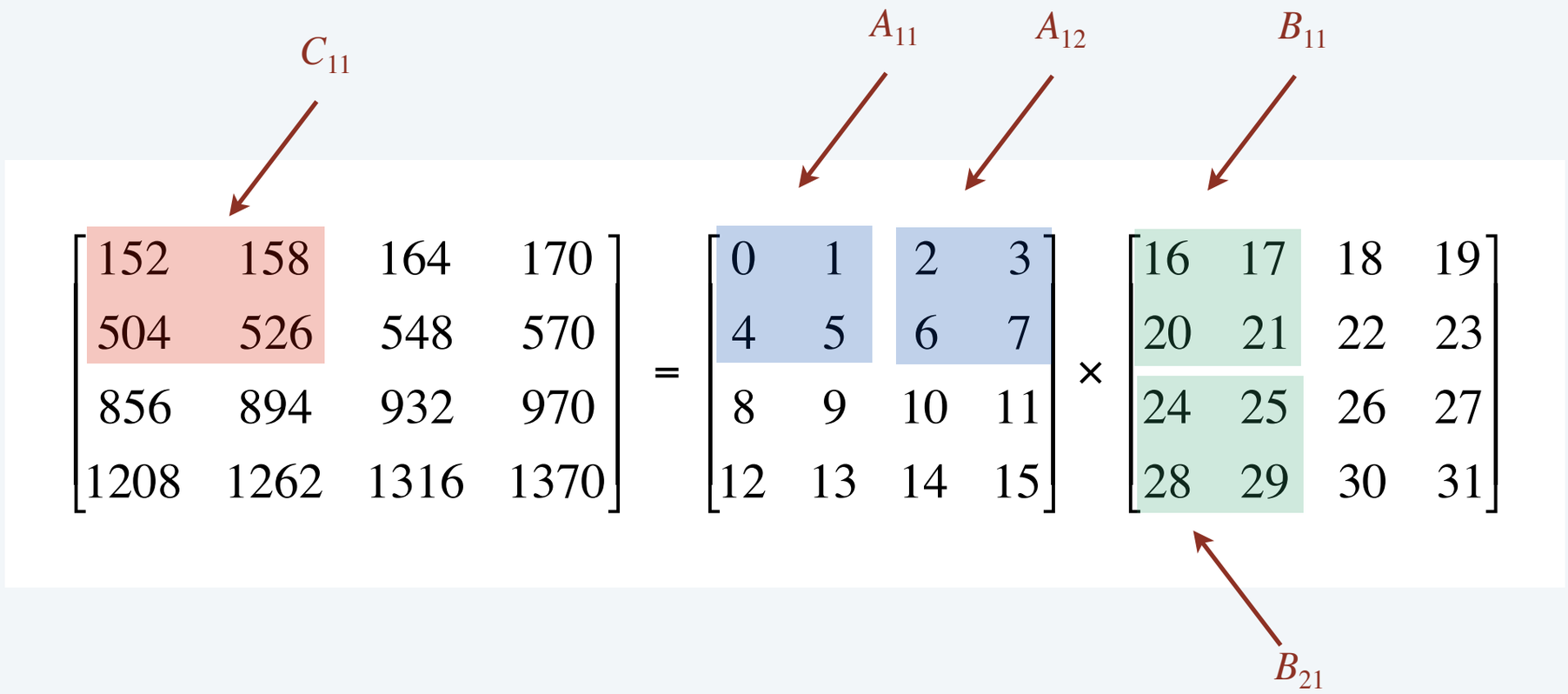
$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q. Is “grade-school” matrix multiplication algorithm asymptotically optimal?

Block matrix multiplication



$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Block matrix multiplication: warmup

To multiply two n-by-n matrices A and B:

- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

n-by-n matrices

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

8 matrix multiplications (of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions (of ½n-by-½n matrices)

Running time. Apply Case 3 of the master theorem.

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Strassen's trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

scalars

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 scalar multiplications

Pf. $C_{12} = P_1 + P_2$

$= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$

$= A_{11} \times B_{12} + A_{12} \times B_{22}$. ✔

Strassen's trick

Key idea. Can multiply two ~~2-by-2~~ *n*-by-*n* matrices via 7 ~~scalar~~ ½*n*-by-½*n* matrix multiplications (plus 11 additions and 7 subtractions).

½*n*-by-½*n matrices*

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 matrix multiplications (of ½n-by-½n matrices)

Pf. $C_{12} = P_1 + P_2$

$= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$

$= A_{11} \times B_{12} + A_{12} \times B_{22}$. ✔

Strassen's algorithm

assume n is a power of 2

STRASSEN(n, A, B)

IF $(n = 1)$ RETURN $A \times B$.

Partition A and B into ½n-by-½n blocks.

$P_1 \leftarrow$ STRASSEN$(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow$ STRASSEN$(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow$ STRASSEN$(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{22}), (B_{11} + B_{22}))$.

$P_6 \leftarrow$ STRASSEN$(n / 2, (A_{12} - A_{22}), (B_{21} + B_{22}))$.

$P_7 \leftarrow$ STRASSEN$(n / 2, (A_{11} - A_{21}), (B_{11} + B_{12}))$.

$7\ T(n / 2) + \Theta(n^2)$

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

$\Theta(n^2)$

RETURN C.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- When n is not a power of 2, pad matrices with zeros to be n'-by-n', where $n \le n' < 2n$ and n' is a power of 2.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

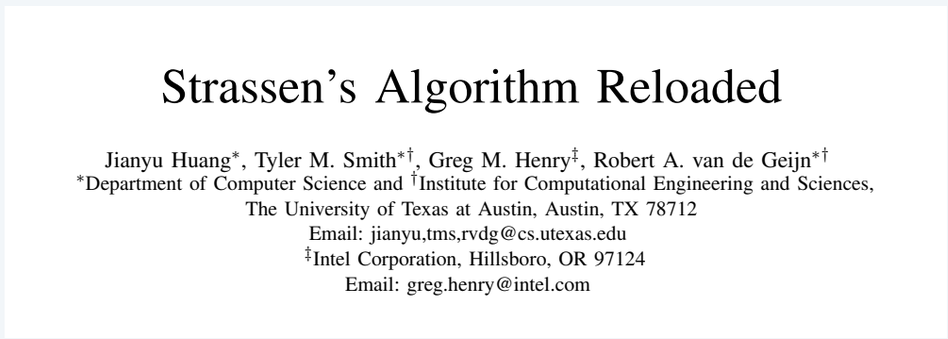
Strassen's algorithm: practice

Implementation issues.

- Sparsity.
- Caching.
- n not a power of 2.
- Numerical stability.
- Non-square matrices.
- Storage for intermediate submatrices.
- Crossover to classical algorithm when n is "small."
- Parallelism for multi-core and many-core architectures.

Common misperception. *"Strassen's algorithm is only a theoretical curiosity."*

- Apple reports 8x speedup when $n \approx 2{,}048$.
- Range of instances where it's useful is a subject of controversy.



Strassen's Algorithm Reloaded

Jianyu Huang*, Tyler M. Smith*†, Greg M. Henry‡, Robert A. van de Geijn*†
*Department of Computer Science and †Institute for Computational Engineering and Sciences,
The University of Texas at Austin, Austin, TX 78712
Email: jianyu,tms,rvdg@cs.utexas.edu
‡Intel Corporation, Hillsboro, OR 97124
Email: greg.henry@intel.com

Divide-and-conquer II: quiz 4

Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n-by-n matrices?

A. $\Theta(n^{\log_3 21})$

B. $\Theta(n^{\log_2 21})$

C. $\Theta(n^{\log_9 21})$

D. $\Theta(n^2)$

Divide-and-conquer II: quiz 5

Is it possible to multiply two 3-by-3 matrices using only 21 scalar multiplications?

A. Yes.

B. No.

C. Unknown.

Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?

A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?

A. Impossible. [Hopcroft–Kerr, Winograd 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Begun, the decimal wars have. [Pan 1978, Bini et al., Schönhage, ...]

- Two 70-by-70 matrices with 143,640 scalar multiplications. $O(n^{2.7962})$
- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$
- A year later. $O(n^{2.7799})$
- December 1979. $O(n^{2.521813})$
- January 1980. $O(n^{2.521801})$

History of arithmetic complexity of matrix multiplication

year	algorithm	arithmetic operations
1858	**“grade school”**	$O(n^3)$
1969	**Strassen**	$O(n^{2.808})$
1978	**Pan**	$O(n^{2.796})$
1979	**Bini**	$O(n^{2.780})$
1981	**Schönhage**	$O(n^{2.522})$
1982	**Romani**	$O(n^{2.517})$
1982	**Coppersmith–Winograd**	$O(n^{2.496})$
1986	**Strassen**	$O(n^{2.479})$
1989	**Coppersmith–Winograd**	$O(n^{2.3755})$
2010	**Strother**	$O(n^{2.3737})$
2011	**Williams**	$O(n^{2.372873})$
2014	**Le Gall**	$O(n^{2.372864})$
		$O(n^{2+\varepsilon})$

galactic algorithms

number of arithmetic operations to multiply two n-by-n matrices

Numeric linear algebra reductions

Matrix multiplication. Given two n-by-n matrices, compute their product.

linear algebra problem	expression	arithmetic complexity
matrix multiplication	$A \times B$	$MM(n)$
matrix squaring	A^2	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$\lvert A \rvert$	$\Theta(MM(n))$
rank	$rank(A)$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L\,U$	$\Theta(MM(n))$
least squares	$\min \lVert Ax - b \rVert_2$	$\Theta(MM(n))$

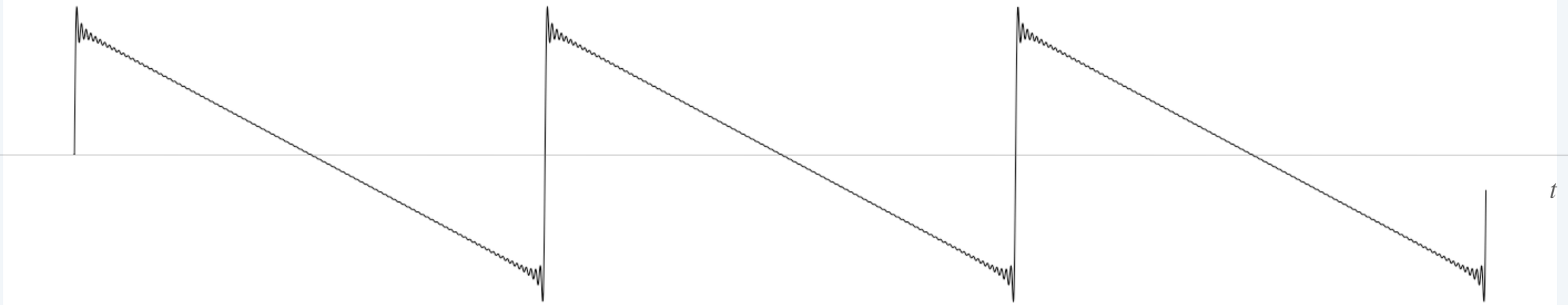
numerical linear algebra problems with the same arithmetic complexity MM(n) as matrix multiplication

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
  - Integer multiplication
  - Matrix multiplication
  - Polynomial multiplication with FFT
- Take-home messages

Fourier analysis

Fourier theorem. [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.



$$y(t) = \frac{2}{\pi} \sum_{k=1}^{n} \frac{\sin kt}{k} \qquad n = 100$$

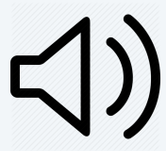
Euler's identity

Euler's identity. $e^{ix} = \cos x + i \sin x$.

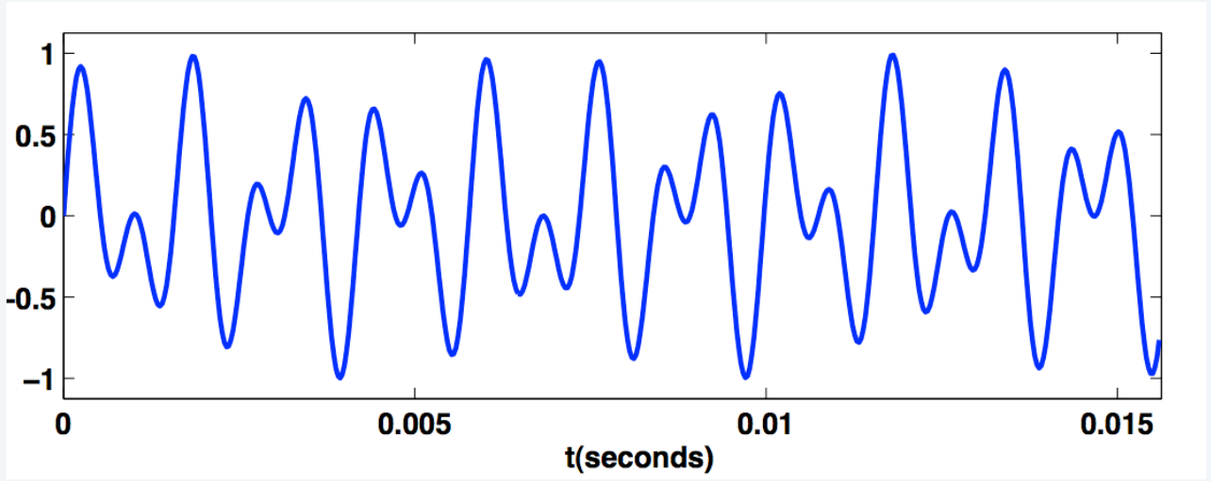
Sinusoids. Sum of sine and cosines = sum of complex exponentials.

Time domain vs. frequency domain

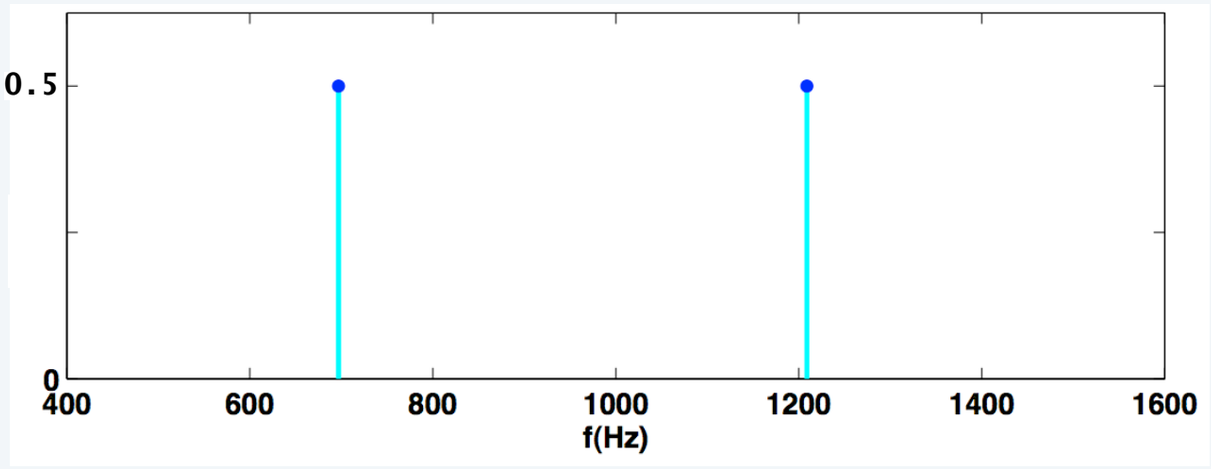
Signal. [touch tone button 1] $y(t) = \frac{1}{2}\sin(2\pi \cdot 697\ t) + \frac{1}{2}\sin(2\pi \cdot 1209\ t)$



Time domain.



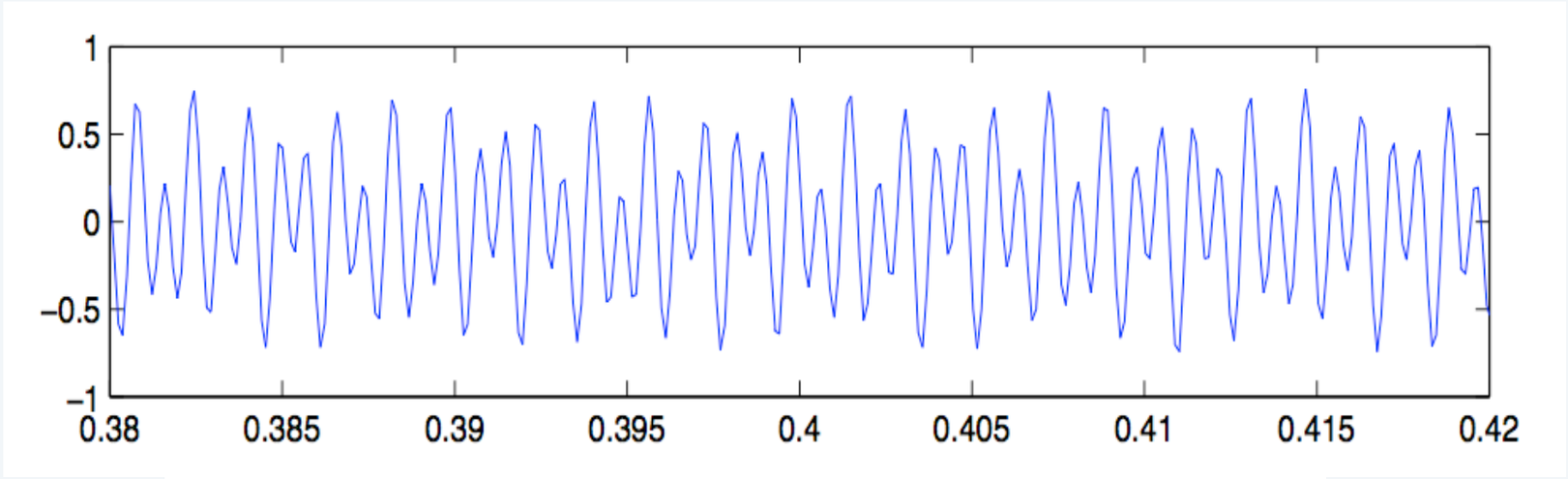
Frequency domain.



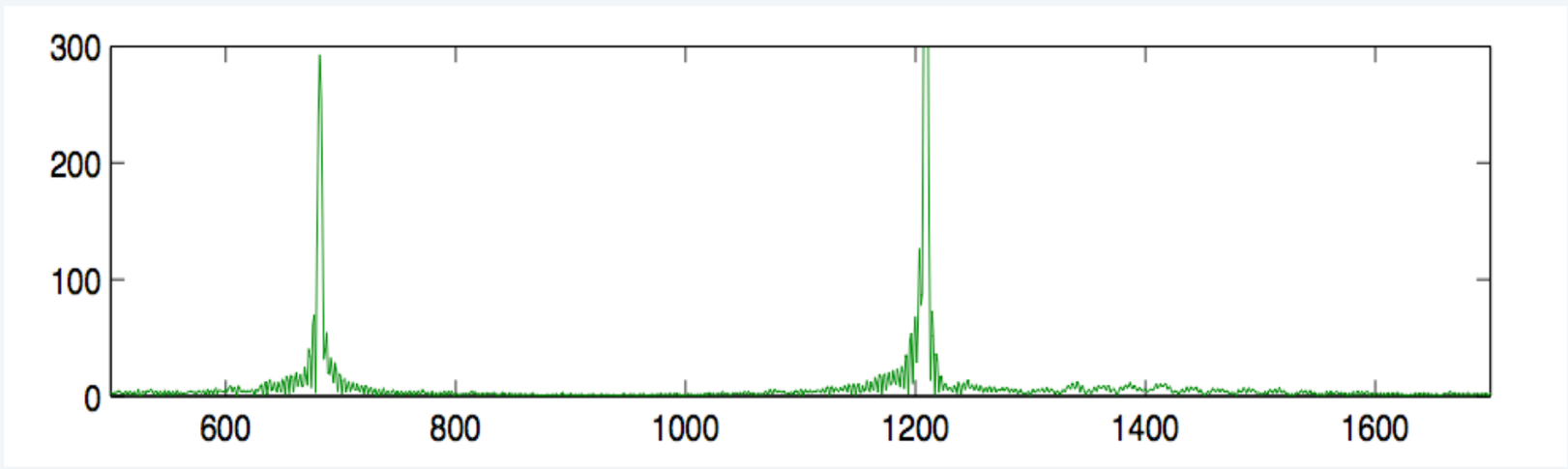
Reference: Cleve Moler, Numerical Computing with MATLAB

Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

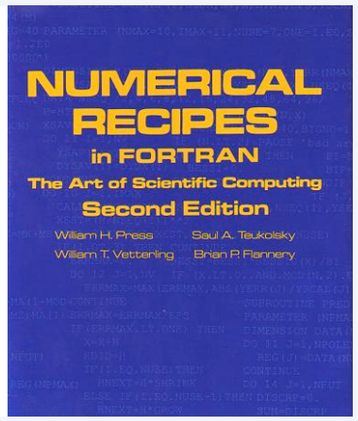
Fast Fourier transform

FFT. Fast way to convert between time domain and frequency domain.

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

> *" If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it. "* — *Numerical Recipes*



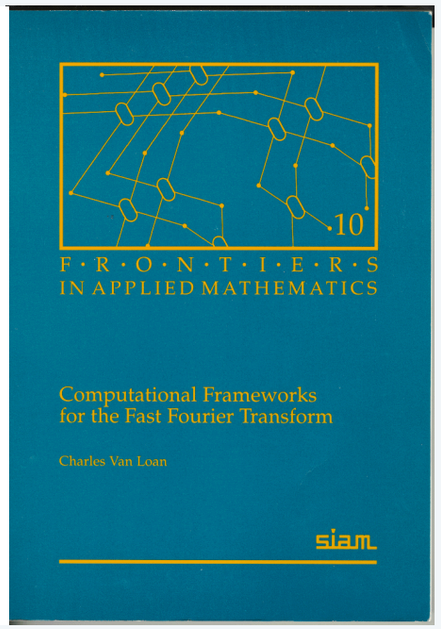
Fast Fourier transform: applications

Applications.

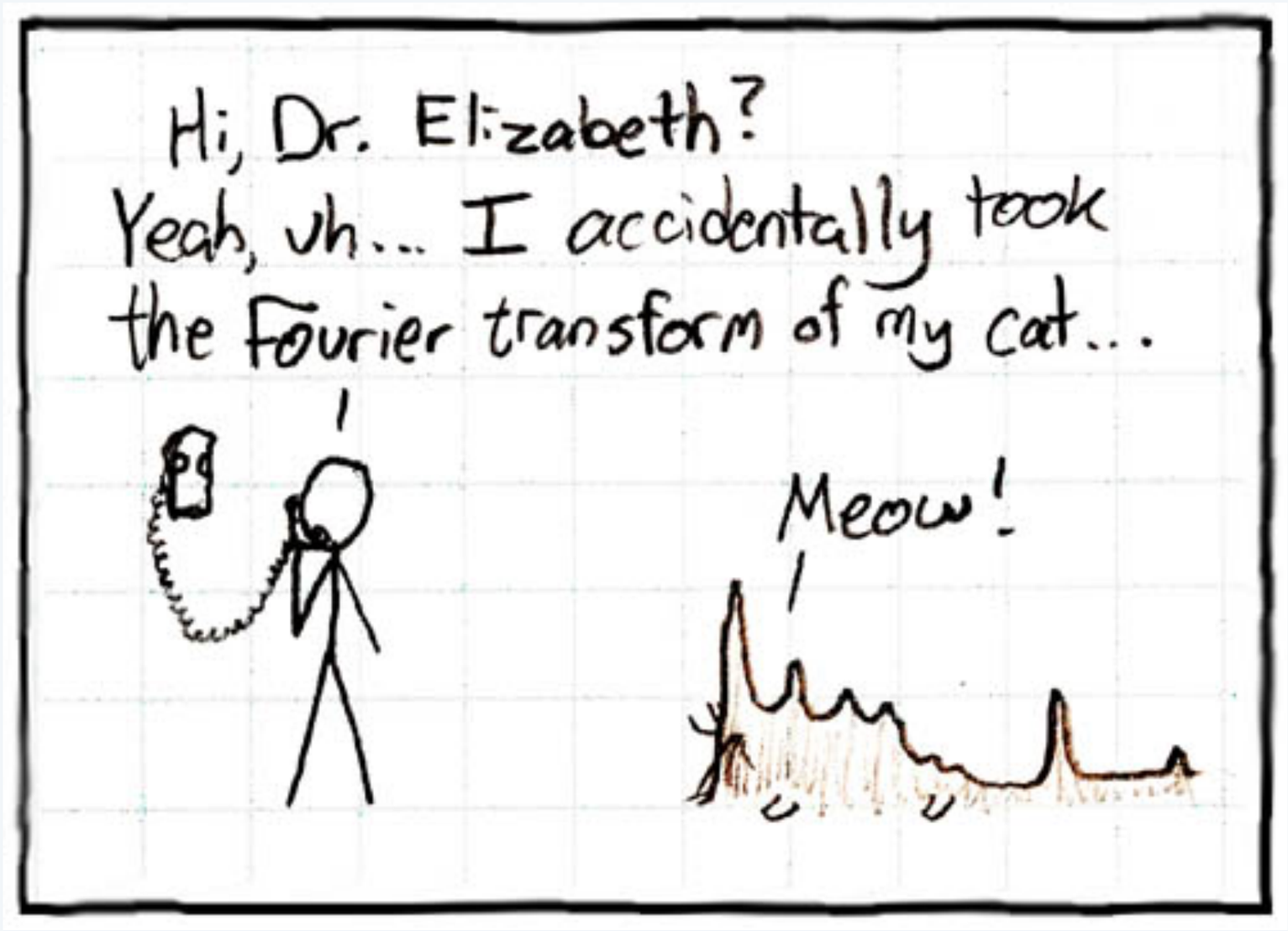
- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

" The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. "

— Charles van Loan



Fast Fourier transform: applications



https://xkcd.com/26

Fast Fourier transform: brief history

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge–König (1924). Laid theoretical groundwork.

Danielson–Lanczos (1942). Efficient algorithm, x-ray crystallography.

Cooley–Tukey (1965). Detect nuclear tests in Soviet Union and track submarines. Rediscovered and popularized FFT.

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2.

Importance not fully realized until emergence of digital computers.

Divide-and-conquer II: quiz 6

What was the subject of Gauss' Ph.D thesis?

A. Gaussian elimination.

B. Fast Fourier transform.

C. Prime number theorem.

D. Cauchy integral theorem.

E. Fundamental theorem of algebra.

F. Angle-preserving maps.

G. Method of least squares.

H. Non-Euclidean geometry.

I. Constructing a regular heptadecagon with straightedge and compass.

A modest Ph.D. dissertation title

DEMONSTRATIO NOVA
THEOREMATIS
OMNEM FVNCTIONEM ALGEBRAICAM
RATIONALEM INTEGRAM
VNIVS VARIABILIS
IN FACTORES REALES PRIMI VEL SECUNDI GRADVS
RESOLVI POSSE

AVCTORE
CAROLO FRIDERICO GAVSS
HELMSTADII
APVD C. G. FLECKEISEN. 1799

1.

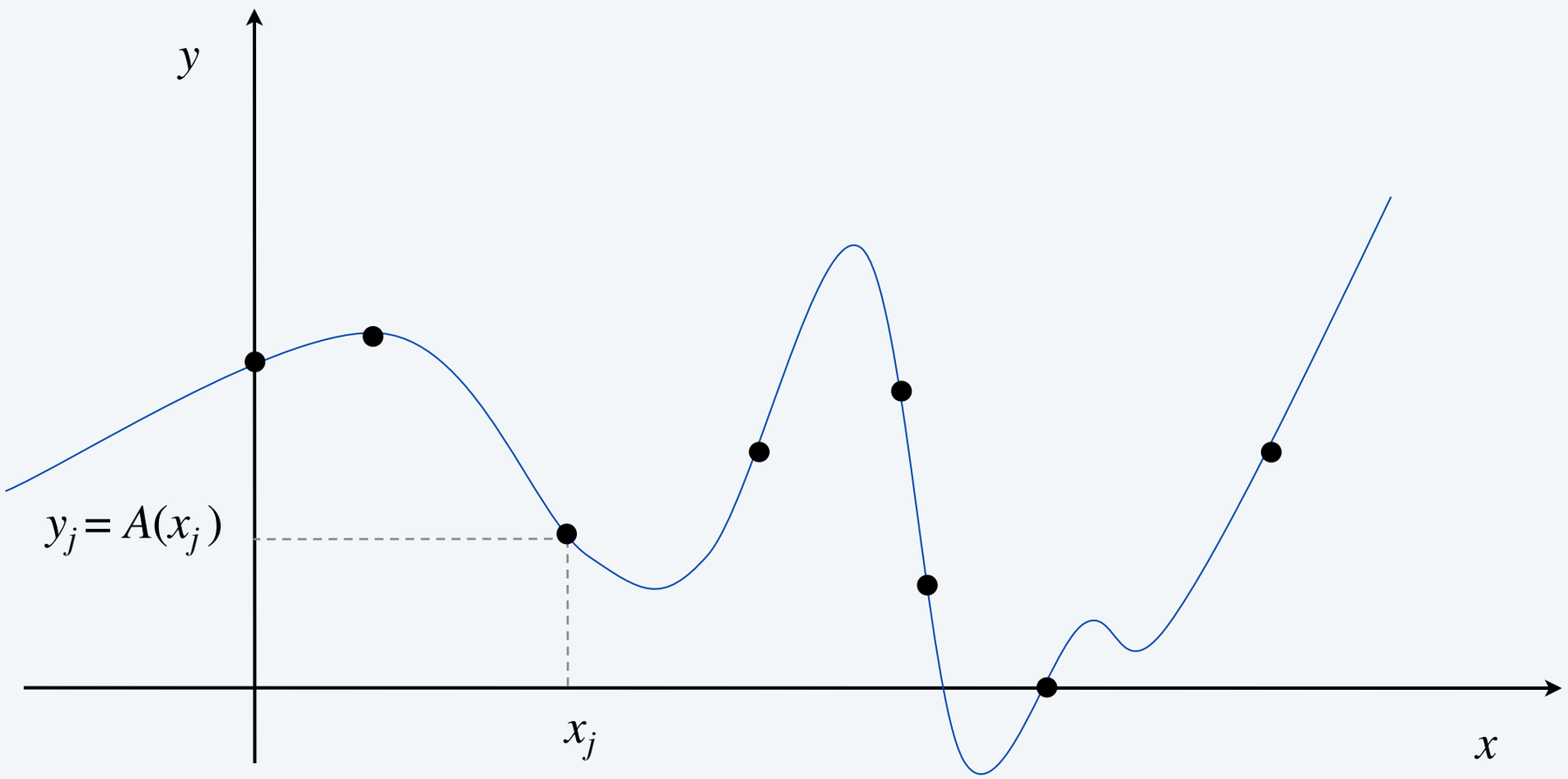
Quaelibet aequatio algebraica determinata reduci potest ad formam $x^m+Ax^{m-1}+Bx^{m-2}+$ etc. $+M=0$, ita vt m sit numerus integer positiuus. Si partem primam huius aequationis per X denotamus, aequationique $X=0$ per plures valores inaequales ipsius x satisfieri supponimus, puta ponendo $x=\alpha$, $x=\beta$, $x=\gamma$ etc. functio X per productum e factoribus $x-\alpha$, $x-\beta$, $x-\gamma$ etc. diuisibilis erit. Vice versa, si productum e pluribus factoribus simplicibus $x-\alpha$, $x-\beta$, $x-\gamma$ etc. functionem X metitur: aequationi $X=0$ satisfiet, aequando ipsam x cuicunque quantitatum α, β, γ etc. Denique si X producto ex m factoribus talibus simplicibus aequalis est (siue omnes diuersi sint, siue quidam ex ipsis identici): alii factores simplices praeter hos functionem X metiri non poterunt. Quamobrem aequatio m^{ti} gradus plures quam m radices habere nequit; simul vero patet, aequationem m^{ti} gradus *pauciores* radices habere posse, etsi X in m factores simplices resolubilis sit:

“ New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree. ”

Polynomials: point-value representation

Fundamental theorem of algebra. A degree n univariate polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree $n-1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



Polynomials: point-value representation

Univariate polynomial. [point-value representation]

$$A(x): \ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

$$B(x): \ (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x): \ (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})$$

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

$$A(x) \times B(x): \ (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluation. $O(n^2)$ using Lagrange's formula.

$$A(x) \ = \ \sum_{k=0}^{n-1} y_k \, \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

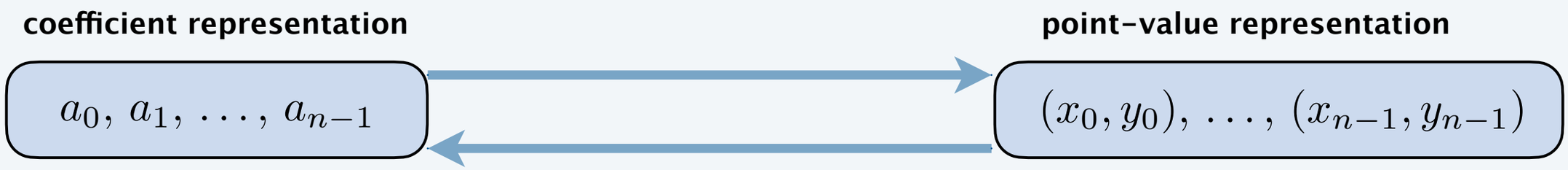
← not used

Converting between two representations

Tradeoff. Either fast evaluation or fast multiplication. We want both!

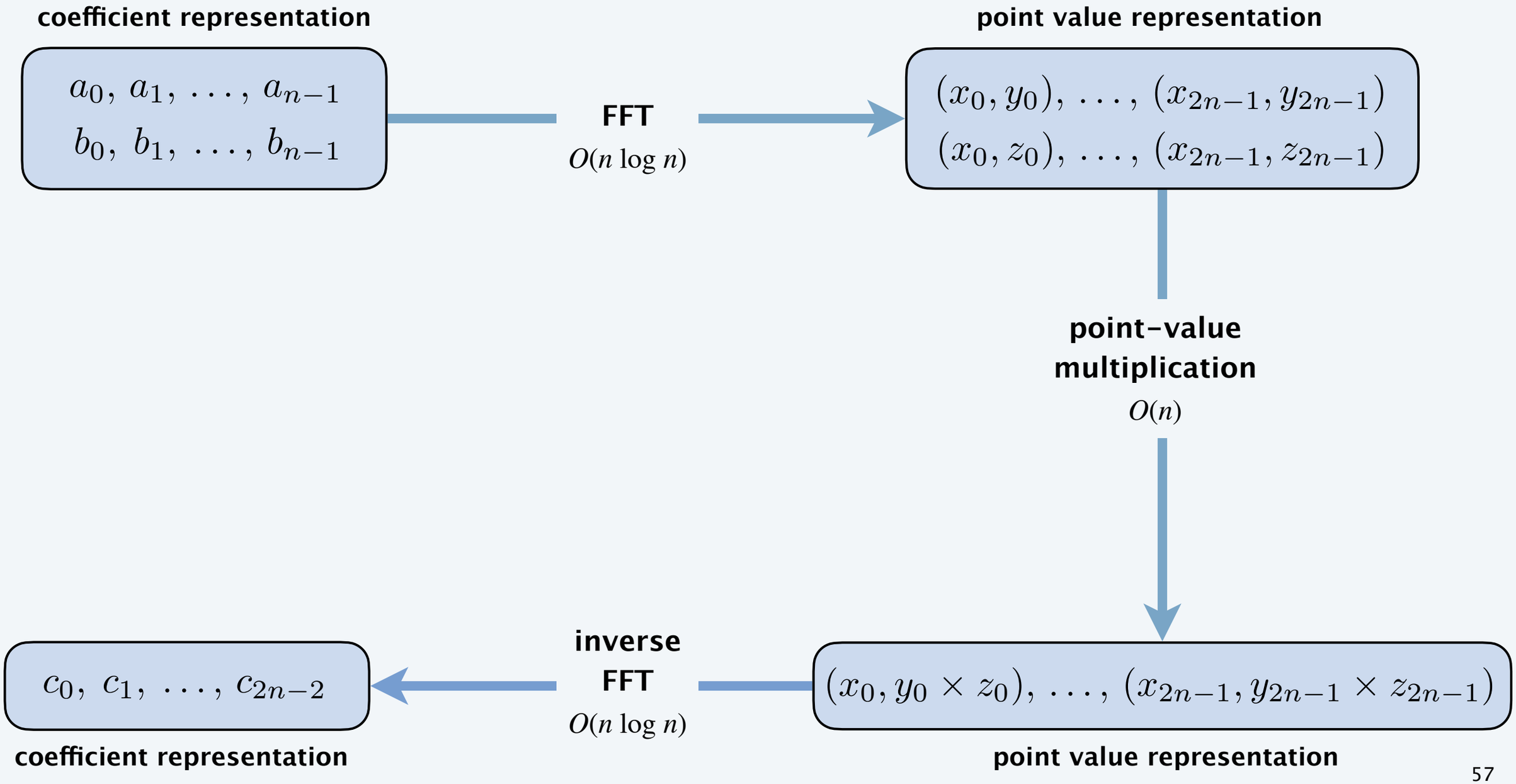
representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations $\Rightarrow$ all ops fast.



Converting between two representations

Application. Polynomial multiplication (coefficient representation).



Converting between two representations: brute force

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Running time. $O(n^2)$ via matrix–vector multiply (or n Horner's).

Converting between two representations: brute force

Point-value ⇒ coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Vandermonde matrix is invertible iff x_i distinct

Running time. $O(n^3)$ via Gaussian elimination.

or $O(n^{2.38})$ via fast matrix multiplication

Divide-and-conquer II: quiz 7

Which divide-and-conquer approach to use to multiply polynomials?

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

A. Divide polynomial into low- and high-degree terms.

$$A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$
$$A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$$

B. Divide polynomial into even- and odd-degree terms.

$$A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$
$$A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

C. Either A or B.

D. Neither A nor B.

Divide-and-conquer

Decimation in time. Divide into even- and odd- degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$

Cooley–Tukey radix 2 FFT

Decimation in frequency. Divide into low- and high-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$
- $A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$
- $A(x) = A_{low}(x) + x^4\, A_{high}(x).$

Sande–Tukey radix 2 FFT

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2).$

Intuition. Choose two points to be ±1.

- $A(1) = A_{even}(1) + 1 A_{odd}(1).$
- $A(-1) = A_{even}(1) - 1 A_{odd}(1).$

← **Can evaluate polynomial of degree n-1 at 2 points by evaluating two polynomials of degree ½n - 1 at only 1 point.**

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2).$

Intuition. Choose four complex points to be $\pm 1, \pm i$.

- $A(1) = A_{even}(1) + 1 A_{odd}(1).$
- $A(-1) = A_{even}(1) - 1 A_{odd}(1).$
- $A(i) = A_{even}(-1) + i A_{odd}(-1).$
- $A(-i) = A_{even}(-1) - i A_{odd}(-1).$

← **Can evaluate polynomial of degree n-1 at 4 points by evaluating two polynomials of degree ½n - 1 at only 2 points.**

Discrete Fourier transform

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Key idea. Choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

$$\underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}}_{y_k = A(\omega^k)} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}}_{\text{Fourier matrix } F_n} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

DFT

Roots of unity

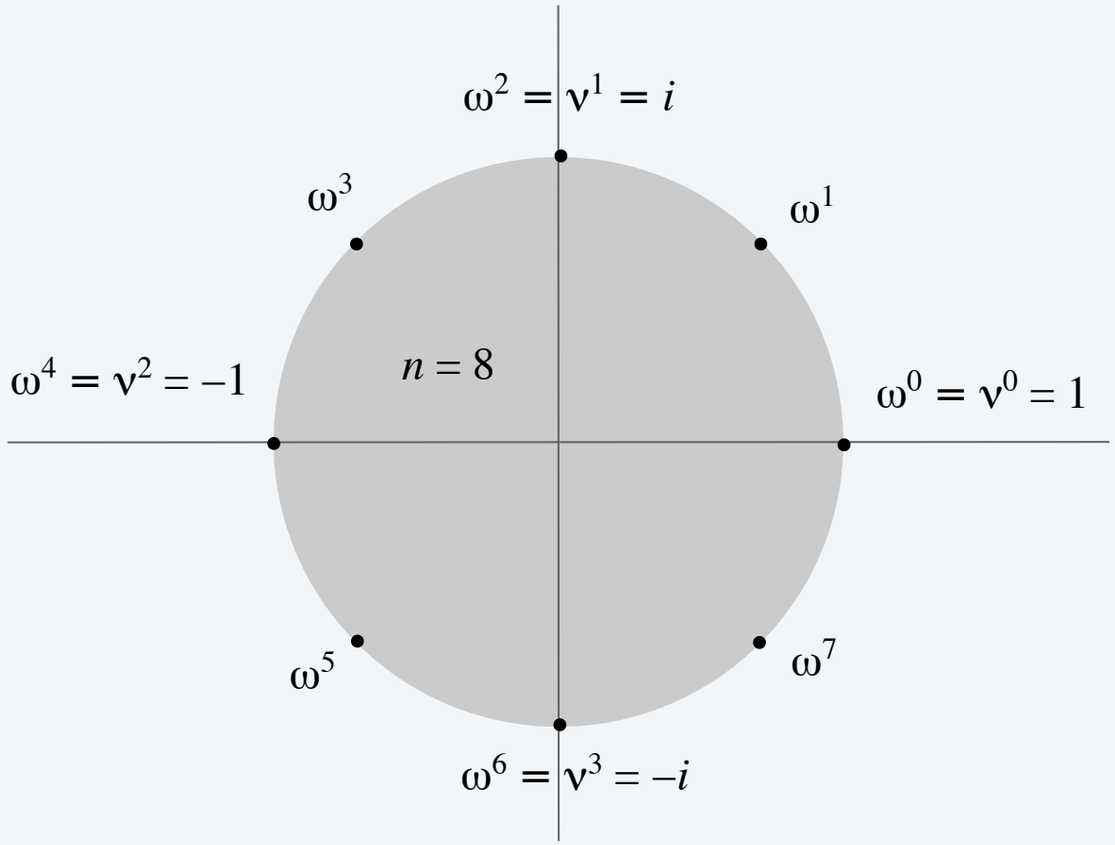
Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

common alternative: $\omega = e^{-2\pi i / n}$

Fact. The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.



Fast Fourier transform

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1}\, x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break up polynomial into even- and odd-degree terms.

- $A_{even}(x) = a_0 + a_2\, x + a_4\, x^2 + \ldots + a_{n-2}\, x^{n/2-1}$.
- $A_{odd}(x) = a_1 + a_3\, x + a_5\, x^2 + \ldots + a_{n-1}\, x^{n/2-1}$.
- $A(x) \quad = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Conquer. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n^{th}$ roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

Combine.

$\nu^k = (\omega^k)^2$

- $y_k \quad = A(\omega^k) \quad = A_{even}(\nu^k) + \omega^k\, A_{odd}(\nu^k), \quad 0 \le k < n/2.$
- $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k\, A_{odd}(\nu^k), \quad 0 \le k < n/2.$

$A(-\omega^k)$

FFT: implementation

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1}\, x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

- $y_k \quad = A(\omega^k) \quad = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2.$
- $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2.$

FFT$(n, a_0, a_1, a_2, \ldots, a_{n-1})$

IF $(n = 1)$ RETURN a_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ FFT$(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ FFT$(n/2, a_1, a_3, a_5, \ldots, a_{n-1})$.

← $2\,T(n/2)$

FOR $k = 0$ TO $n/2 - 1$.

$\omega^k \leftarrow e^{2\pi i k/n}$.

$y_k \leftarrow e_k + \omega^k d_k$.

$y_{k+n/2} \leftarrow e_k - \omega^k d_k$.

← $\Theta(n)$

RETURN $(y_0, y_1, y_2, \ldots, y_{n-1})$.

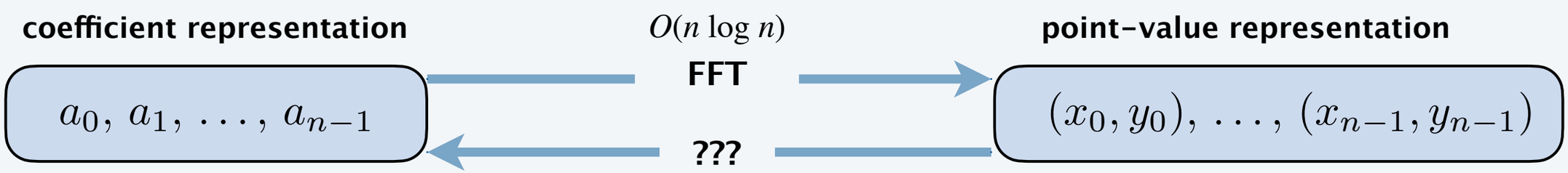
FFT: summary

Theorem. The FFT algorithm evaluates a degree $n - 1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations and $O(n)$ extra space.

assumes n is a power of 2

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



Divide-and-conquer II: quiz 8

When computing the FFT of $(a_0, a_1, a_2, \ldots, a_7)$, which are the first two coefficients involved in an arithmetic operation?

A. a_0 and a_1.

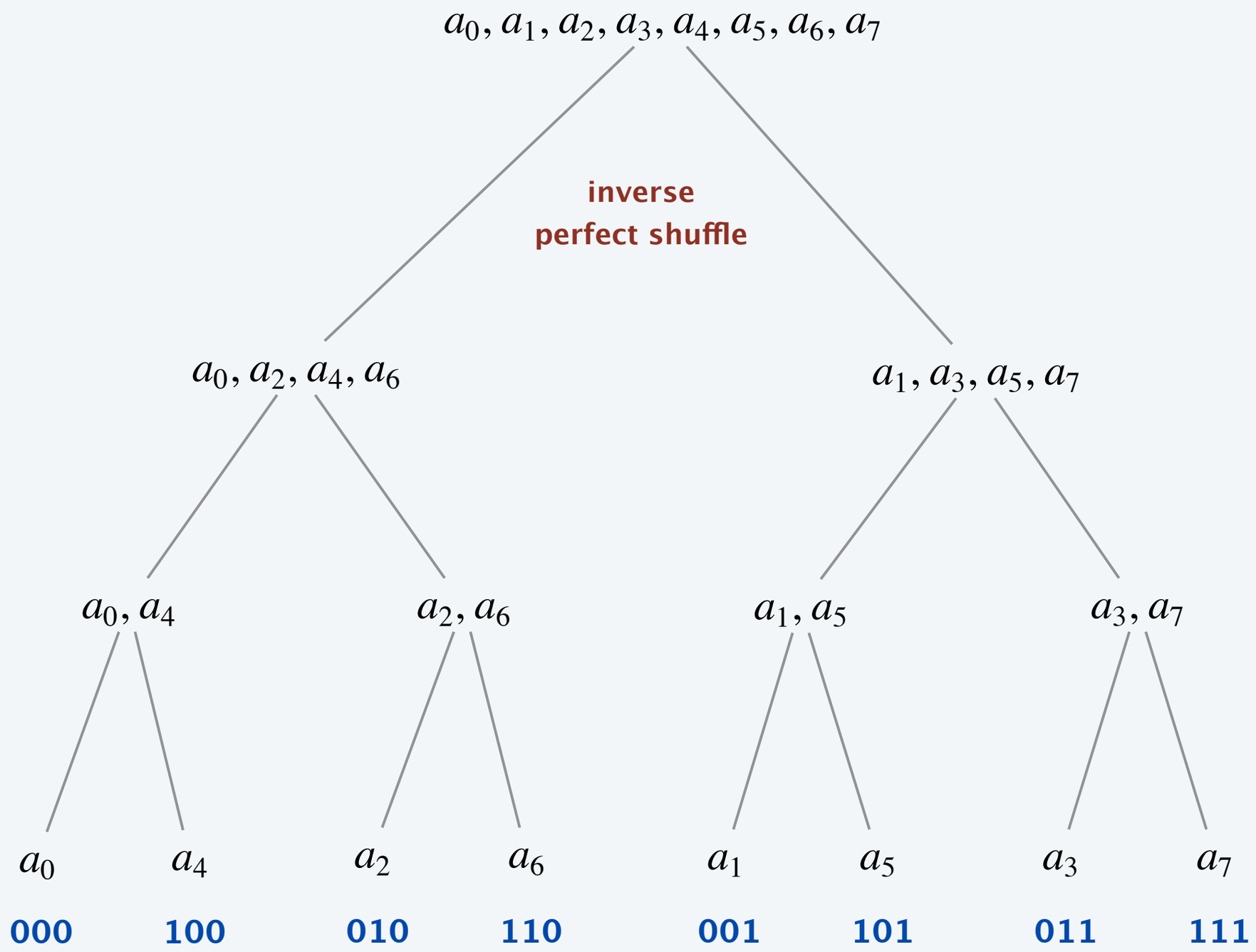
B. a_0 and a_2.

C. a_0 and a_4.

D. a_0 and a_7.

E. None of the above.

FFT: recursion tree



"bit-reversed" order

FFT: Fourier matrix decomposition

Alternative viewpoint. FFT is a recursive decomposition of Fourier matrix.

Fourier matrix

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \qquad D_n = \begin{bmatrix} \omega^0 & 0 & 0 & \dots & 0 \\ 0 & \omega^1 & 0 & \dots & 0 \\ 0 & 0 & \omega^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \omega^{n-1} \end{bmatrix}$$

DFT

$$y = F_n\, a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2}\, a_{even} \\ F_{n/2}\, a_{odd} \end{bmatrix}$$

Inverse discrete Fourier transform

Point-value ⇒ coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Inverse DFT

Fourier matrix inverse $(F_n)^{-1}$

Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix F_n is given by following formula:

$$G_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

F_n / $\sqrt{n}$ is a unitary matrix

Consequence. To compute the inverse FFT, apply the same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide the result by n).

Inverse FFT: proof of correctness

Claim. F_n and G_n are inverses.

Pf.

$$(F_n G_n)_{kk'} = \frac{1}{n}\sum_{j=0}^{n-1} \omega^{kj}\omega^{-jk'} = \frac{1}{n}\sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma (below)

Summation lemma. Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

Pf.

- If k is a multiple of n, then $\omega^k = 1 \Rightarrow$ series sums to n.
- Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1})$.
- if $\omega^k \neq 1$, then $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0 \Rightarrow$ series sums to 0. ▪

Inverse FFT: implementation

Note. Need to divide result by n.

INVERSE-FFT$(n, y_0, y_1, y_2, \ldots, y_{n-1})$

IF $(n = 1)$ RETURN y_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ INVERSE-FFT$(n / 2, y_0, y_2, y_4, \ldots, y_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ INVERSE-FFT$(n / 2, y_1, y_3, y_5, \ldots, y_{n-1})$.

FOR $k = 0$ TO $n / 2 - 1$.

$\omega^k \leftarrow e^{-2\pi i k / n}$.

$a_k \leftarrow e_k + \omega^k d_k$.

$a_{k + n/2} \leftarrow e_k - \omega^k d_k$.

RETURN $(a_0, a_1, a_2, \ldots, a_{n-1})$.

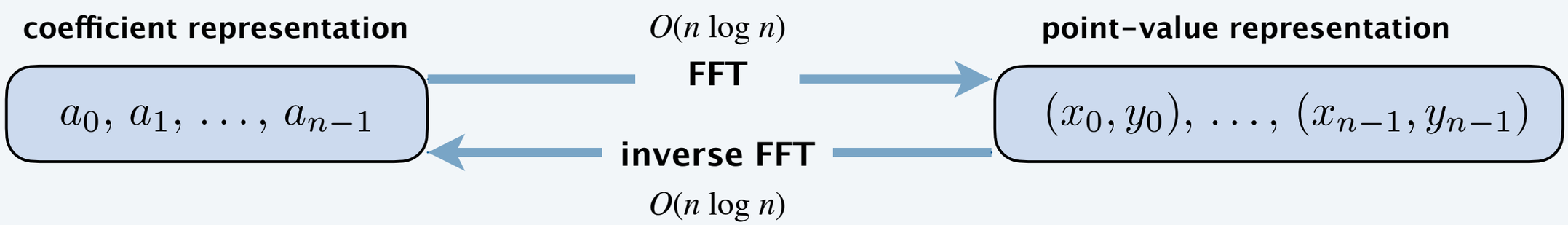
switch roles of a_i and y_i

Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations.

assumes n is a power of 2

Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.

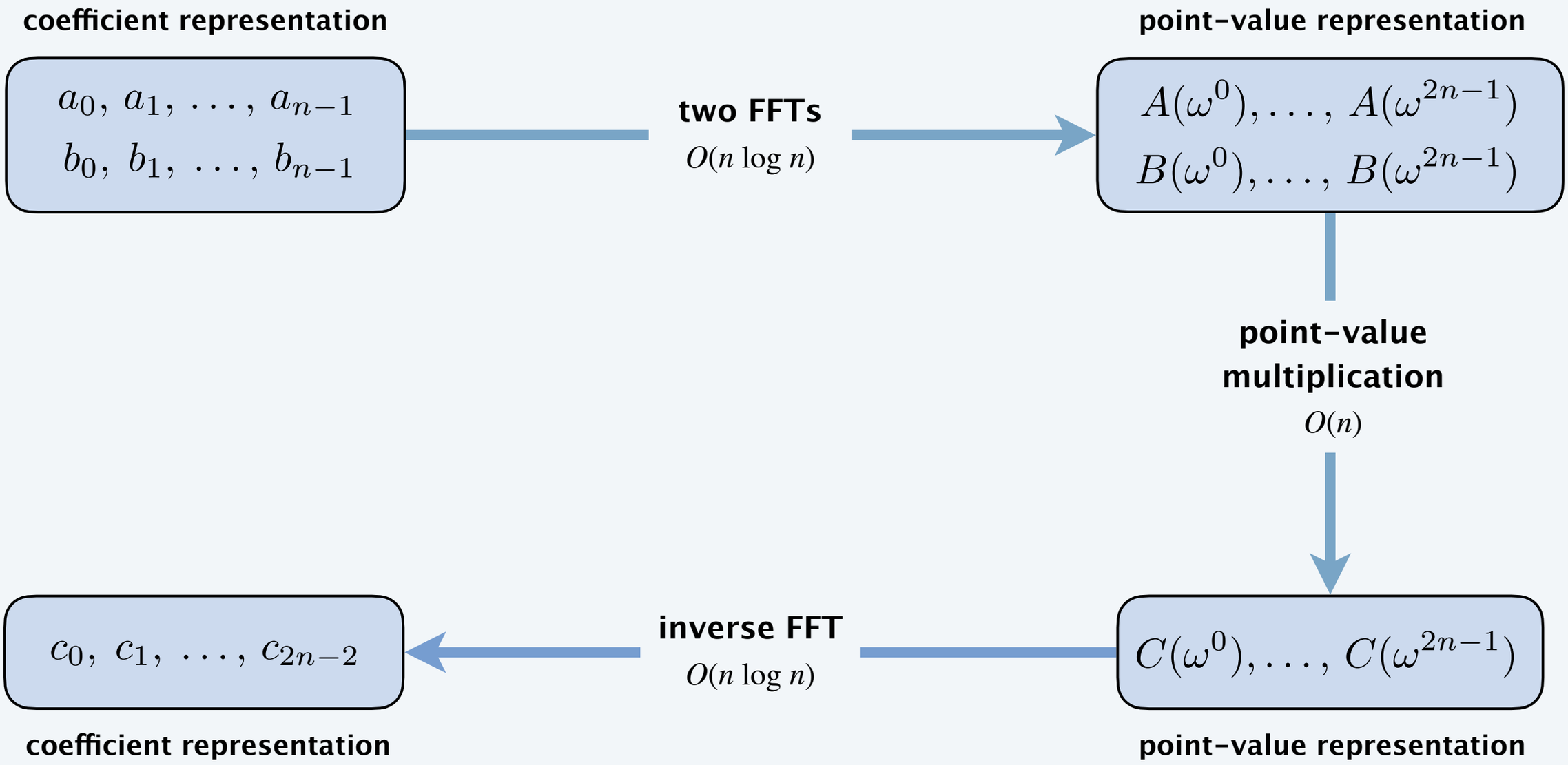


Polynomial multiplication

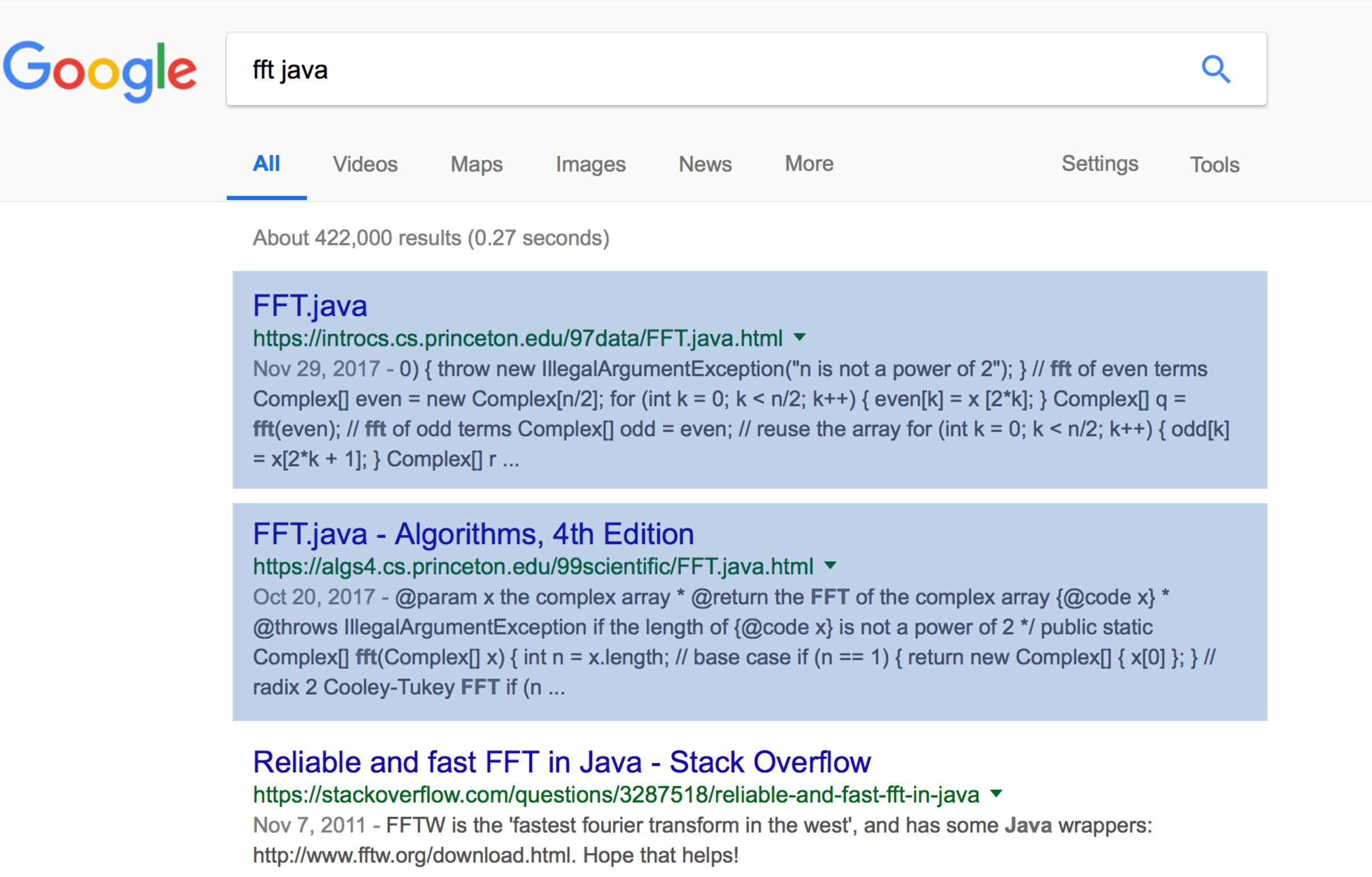
Theorem. Given two polynomials $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ and $B(x) = b_0 + b_1 x + \ldots + b_{n-1} b^{n-1}$ of degree $n - 1$, can multiply them in $O(n \log n)$ arithmetic operations.

pad with 0s to make n a power of 2

Pf.



FFT in practice ?



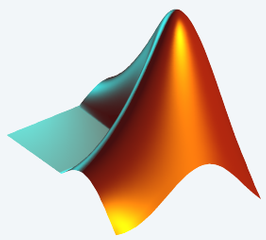
FFT in practice

Fastest Fourier transform in the West. [Frigo–Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Core algorithm is an in-place, nonrecursive version of Cooley–Tukey.
- Instead of executing a fixed algorithm, it evaluates the hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Runs in $O(n \log n)$ time, even when n is prime.
- Multidimensional FFTs.
- Parallelism.



FFTW

http://www.fftw.org

Top 10 algorithms of the 20th century

FROM THE EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-in-Chief

THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED "THE TOP 10 ALGORITHMS OF THE CENTURY."

Daniel Rockmore describes the FFT as an algorithm "the whole family can use." The FFT is perhaps the most ubiquitous algorithm in use today to analyze and manipulate digital or discrete data. The FFT takes the operation count for discrete Fourier transform from $O(N^2)$ to $O(N \log N)$.

Integer multiplication, redux

Integer multiplication. Given two n-bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

Convolution algorithm.

- Form two polynomials.
- Note: $a = A(2), b = B(2)$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ floating-point operations.

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$$

Theory. [Schönhage–Strassen 1971]

- $O(n \log^2 n)$ bit operations. ← FFT over complex numbers; need $O(\log n)$ bits of precision
- $O(n \log n \cdot \log \log n)$ bit operations. ← FFT over ring of integers (modulo a Fermat number)

Practice. [GNU Multiple Precision Arithmetic Library]

Switches to FFT-based algorithm when n is large (≥ 5–10K).

3-SUM (REVISITED)

3-SUM. Given three sets X, Y, and Z of n integers each, determine whether there is a triple $i \in X$, $j \in Y$, $k \in Z$ such that $i + j = k$.

Assumption. All integers are between 0 and m.

Goal. $O(m \log m + n \log n)$ time.

$m = 19,\ n = 3$

$X = \{\ 4, 7, 10\ \}$

$Y = \{\ 5, 8, 15\ \}$

$Z = \{\ 4, 13, 19\ \}$

a yes instance
(4 + 15 = 19)

3-SUM (REVISITED)

An $O(m \log m + n)$ solution.

- Form polynomial $A(x) = a_0 + a_1 x + \ldots + a_m x^m$ with $a_i = 1$ iff $i \in X$.
- Form polynomial $B(x) = b_0 + b_1 x + \ldots + b_m x^m$ with $b_j = 1$ iff $j \in Y$.
- Compute product/convolution $C(x) = A(x) \times B(x)$.
- The coefficient c_k = number of ways to choose an integer $i \in X$ and an integer $j \in Y$ that sum to exactly k.
- For each $k \in Z$: check whether $c_k > 0$.

$m = 19,\ n = 3$

$X = \{\ 4, 7, 10\ \}$

$Y = \{\ 5, 8, 15\ \}$

$Z = \{\ 4, 13, 19\ \}$

a yes instance
(4 + 15 = 19)

$$A(x) = x^4 + x^7 + x^{10}$$

$$B(x) = x^5 + x^8 + x^{15}$$

$$C(x) = x^9 + 2x^{12} + 2x^{15} + x^{18} + \boxed{x^{19}} + x^{22} + x^{25}$$

Divide and Conquer

- Introduction
- Closest pair of points
- Solving recurrent equations
- Multiplication problems
  - Integer multiplication
  - Matrix multiplication
  - Polynomial multiplication with FFT
- Take-home messages

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into two subproblems of size $n/2$. ← $O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. ← $O(n)$ time

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $O(n \log n)$.

Common situation.
Not always the case.

attributed to Julius Caesar

Thanks for your attention!
Discussions?

Reference

https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/05DivideAndConquerI.pdf

https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/05DivideAndConquerII.pdf

Algorithms by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani: Chap. 2.

Algorithm design: Chap. 5.

Introduction to algorithms (4th edition): Chap. 4.